Universidade de Aveiro Departamento de Matemática
2010

Aurélio de Jesus Correia Barbosa Vicente

Métodos de Aproximação Numérica usando o Matlab

Universidade de Aveiro Departamento de Matemática
2010

Aurélio de Jesus Correia Barbosa Vicente

Métodos de Aproximação Numérica usando o Matlab

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Matemática Aplicada à Engenharia, realizada sob a orientação científica do Prof. Doutor António Jorge Monteiro Neves, Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro

Apoio Financeiro do Instituto Português de Apoio ao Desenvolvimento (IPAD)

“Todas as ciências exactas são dominadas pela ideia da aproximação.”

Bertrand Russel

o júri

Presidente	Prof. Doutor Domingos Moreira Cardoso Professor Catedrático do Departamento de Matemática da Universidade de Aveiro
Arguente	Prof. Doutor Gastão Silves Ferreira Frederico Professor Auxiliar do Departamento de Ciência e Tecnologia da Universidade de Cabo Verde
Orientador	Prof. Doutor António Jorge Monteiro Neves Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro

Agradecimentos

Agradeço ao meu orientador, o Prof. Doutor António Jorge Neves pelo incentivo e apoio dado na elaboração dessa dissertação. A visão científica acerca do tema, bem como as sugestões foram de uma importância enorme.

Agradeço ainda à minha esposa e filho pela compreensão quando me punha em isolamento a fim de conseguir a concentração necessária para a elaboração da dissertação.

Palavras-chave Aproximação numérica, interpolação polinomial, Splines, Curvas de Bézier, MatLab.

Resumo

Neste trabalho são abordados métodos de aproximação numérica baseados em interpolação polinomial, interpolação segmentada usando Splines e curvas de Bézier. Apresenta-se uma descrição teórica desses métodos, estuda-se a sua aplicabilidade e compara-se a qualidade das aproximações obtidas, quando os mesmos são aplicados a exemplos práticos que vão sendo introduzidos ao longo do texto. Para o efeito, recorre-se também à experimentação numérica usando-se rotinas apropriadas em MatLab (MATrix LABoratory), as quais ajudam nas análises comparativas das vantagens e desvantagens dos diferentes métodos estudados.

Keywords Numerical approximation, polynomial interpolation, Splines, Bezier curves, MatLab.

Abstract

In this work are studied some methods of numerical approximation based on polynomial interpolation, interpolation using splines and Bezier curves.

A theoretical description of these methods are presented, we study their applicability and compares the quality of approximations obtained when they are applied to practical examples, which are being introduced throughout the text. For this purpose, it is also presented the numerical experiments using appropriet functions in MatLab (MATrix LABoratory programming language), which help in the comparative analysis of advantages and disadvantages of different methods.

Conteúdos

1. Introdução

O problema do cálculo de quantidades aproximadas é um problema enfrentado pelos matemáticos desde a antiguidade. Como exemplo pode-se referir a fórmula de aproximação da raíz quadrada de um número, atribuída aos Babilónios (Karl, 2006).

Contudo, a aproximação numérica é um ramo da matemática relativamente jovem, que faz uso do conceito da dependência entre quantidades, ou seja, do conceito de funções. A primeira abordagem para a definição de uma função com base na dependência entre quantidades, resumida numa determinada fórmula, foi desenvolvida por Euler[1] (1707-1783).

No entanto, desde esses tempos até aos dias de hoje e, com a evolução dos computadores, os métodos de aproximação numérica foram objecto de grandes desenvolvimentos, muitos deles resultantes da sua aplicação nas mais variadas áreas das ciências e engenharias. Muitos dos problemas matemáticos suscitados nessas áreas só podem ser resolvidos com recurso a métodos numéricos, não sendo possível obter soluções exactas, interessa, pois, saber quais as vantagens e desvantagens de alguns desses métodos e como utilizá-los computacionalmente, por forma a conseguirem obter-se soluções fiáveis para aqueles problemas.

Com este trabalho pretendem-se abordar métodos de aproximação numérica, tirando partido da ferramenta computacional Matlab (MATrix LABoratory) para o desenvolvimento de rotinas que possibilitem a aplicação desses métodos, de forma a poder-se comparar a sua aplicabilidade e a analisarem-se as suas vantagens/desvantagens.
Assim, serão estudados métodos de aproximação numérica baseados em interpolação polinomial, interpolação segmentada usando Splines e curvas de Bézier. Apresenta-se uma

[1] Leonhard Euler, Matemático e Físico Suíço.

descrição teórica desses métodos, averigua-se a sua aplicabilidade e discutem-se as vantagens e desvantagens, muitas vezes recorrendo a exemplos práticos com simulações e experimentações numéricas utilizando o *software* MatLab.

1.1. Métodos de aproximação numérica

A aproximação numérica envolve a construção de métodos numéricos e o desenvolvimento de algoritmos que os implementem de modo a calcularem-se soluções aproximadas de problemas matemáticos para os quais não é possível obter soluções analíticas.

Podemos definir algoritmo como uma sequência ordenada e finita de operações bem definidas e eficazes que, quando executadas por um computador terminam sempre num determinado período de tempo, produzindo uma solução ou indicando que tal solução não pode ser obtida.

O nosso interesse por este tema surge no sentido em que, como se sabe, muitos problemas em ciências e engenharia envolvem modelos matemáticos para os quais não existem métodos exactos de resolução, sendo necessário recorrer à utilização de métodos numéricos que conduzem a soluções aproximadas.
Pretende-se que tais soluções aproximadas sejam fiáveis, ou seja, que o erro que lhes está associado esteja limitado dentro de valores aceitáveis, consoante a precisão requerida para os resultados. Muitas vezes, para se conseguir este objectivo é necessária a experimentação numérica de vários métodos, por forma a compararem-se os resultados obtidos por esses métodos.

Com o presente trabalho, pretende-se estudar métodos de aproximação numérica adequados à resolução de determinados problemas e apresentar algoritmos de alguns desses métodos usando-se como ferramenta computacional o MatLab. Com esta ferramenta, interactiva e de alta performance, orientada à execução de tarefas que envolvam cálculo numérico (Morais, 2006), pretende-se desenvolver rotinas (funções em MatLab) para experimentação e simulação numérica de alguns dos métodos estudados.
Esses métodos serão aplicados a exemplos práticos concretos, sendo feitas comparações relativamente a vantagens e desvantagens de cada um deles.

1.2. Etapas na resolução numérica de um problema

Os métodos numéricos permitem dar resposta a problemas matemáticos que não possuem soluções analíticas. Para estes casos, obtém-se uma solução numérica que, em caso de convergência, aproxima a solução exacta do problema.

Deste modo, para a obtenção de uma solução aproximada de um dado problema real, torna-se necessário percorrer as seguintes etapas:

- Definição e recolha de dados do problema real;
- Modelação matemática do problema;
- Escolha adequada do método numérico a usar;
- Cálculo da solução do problema através da computação de um algoritmo numérico;
- Análise da fiabilidade dos resultados (erro cometido) e interpretação dos mesmos;

Na escolha do método, deve ter-se em conta os seguintes pressupostos:

- Precisão desejada para os resultados;
- Convergência do método, ou seja, garantia de que o método quando aplicado permite obter uma solução aproximada que tende (tanto quanto possível) para a solução do problema;
- Esforço computacional dispendido no cálculo da solução do problema;

Para a obtenção da solução numérica do problema, com uma razoável precisão deve utilizar-se um algoritmo numérico adequado e uma ferramenta computacional com razoável poder de cálculo científico, como é o caso do programa MatLab que aqui vai ser usado.

1.3. Estrutura da dissertação

Quanto à estruturação do trabalho desta dissertação, após a presente introdução, no capítulo 2, faz-se uma abordagem teórico-prática a métodos de interpolação polinomial, nomeadamente, interpolação de Lagrange, fórmula interpoladora de Newton e interpolação de Hermite, comparando-se estas técnicas de aproximação numérica em vários exemplos,

acompanhados de programas em MatLab que ilustram o potencial destes métodos e permitem concluir sobre algumas das vantagens e desvantagens dos mesmos. Estuda-se também a minimização do erro de interpolação cometido através da escolha adequada dos nós de interpolação.

No capítulo 3, apresentam-se métodos de aproximação numérica baseados em interpolação segmentada (por troços), neste caso, os Splines e as curvas de Bézier, os quais permitem construir aproximações de melhor qualidade quando se verifica uma variabilidade considerável no comportamento dos dados a aproximar (quer no caso discreto, quer no caso contínuo, com a substituição de uma função complicada por um polinómio aproximante mais simples). Com estes métodos mostra-se, com recurso ao MatLab, como é possível construirem-se curvas suaves capazes de produzir razoáveis aproximações nas mais variadas aplicações, tais como, no design de automóveis, no traçado de símbolos em design gráfico e mesmo no caso de contornos fechados (incluindo laços). A este respeito são feitas as descrições teóricas dos Splines, Splines na forma paramétrica e B-Splines, assim como, a metodologia de construção das curvas de Bézier. Em todos estes métodos dá-se enfâse à construção de aproximações numéricas cúbicas (de grau três), pois, estas conseguem apresentar bons desempenhos nas aplicações onde são usadas, não sendo necessário um trabalho, deveras mais moroso, para se obterem aproximações de grau mais elevado. Ainda no capítulo 3, faz-se a comparação entre os vários métodos aí estudados aplicando estes na construção de modelações geométricas em casos práticos, a fim de se identificarem os pontos fortes e fracos de cada um deles.

Finalmente, no capítulo 4, faz-se uma breve referência à extensão da interpolação polinomial ao caso bidimensional na forma de Lagrange, apresentando-se simulações adequadas em MatLab, que ilucidam como se pode proceder para se obterem gráficos de aproximações de funções f no espaço bidimensional (de funções de duas variáveis x e y, $z=f(x,y)$).

2. Métodos de Interpolação Polinomial

Interpolar uma função $f(x)$ consiste em aproximar essa função por uma outra função $\Phi(x)$, escolhida de entre uma classe de funções definidas *a priori* e que satisfaçam algumas propriedades. A função $\Phi(x)$ é, então, usada em detrimento da função $f(x)$.

A necessidade de se efectuar a substituição da função $f(x)$ pela função $\Phi(x)$, surge em várias situações práticas, como por exemplo:

- Quando são conhecidos somente os valores numéricos da função $f(x)$ para um conjunto de pontos e é necessário calcular o valor da função num ponto não conhecido (caso discreto);
- Quando a função em estudo tem uma expressão complicada, para qual, por exemplo, operações como a diferenciação e a integração são difíceis de serem realizadas (caso contínuo).

Dados $n+1$ pares de pontos distintos (x_i, y_i), o problema da aproximação consiste em encontrar uma função $y = \Phi(x)$, tal que

$$\Phi(x_i) = y_i \text{ para } i = 0,1,...n \tag{2.1}$$

onde y_i são valores dados a priori. Diz-se então que Φ interpola os valores $\{y_i\}_{i=0,1,2,...,n}$ nos nós distintos $\{x_i\}_{i=0,1,2,...,n}$ (Quarteroni, 2007).

Definição 1. Designa-se por interpolação polinomial, se Φ é um polinómio algébrico, por interpolação trigonométrica, se Φ for um polinómio trigonométrico e por interpolação polinomial segmentada (ou interpolação com Splines), se Φ for sectorialmente um polinómio.

No entanto, neste trabalho, vamos apenas tratar os casos de interpolação polinomial e interpolação segmentada.

Definição 2. Dados $n+1$ pontos distintos $x_0, x_1, ..., x_n$, num intervalo arbitrário $[a,b]$, e os respectivos valores de uma função f nesses pontos, $f(x_0), f(x_1), ..., f(x_n)$, a interpolação polinomial consiste em encontrar um polinómio $P_n(x)$, de grau menor ou igual a n, que coincide com f nos $n+1$ pontos dados. Ou seja, $p_n(x_i) = a_n x_i^n + ... + a_1 x_i + a_0 = f(x_i), \quad i = 0,1,...,n$.

Os pontos x_i são designados de nós de interpolação e os pontos $y_i = f(x_i)$ de valores nodais.

O polinómio $P_n(x)$ é designado por polinómio interpolador da função f naquele suporte de $n+1$ pontos e pode ser utilizado para calcular uma estimativa do valor da função f num dado ponto x diferente de x_i no intervalo $[a,b]$.

Assim, a função $f(x)$ é aproximada pelo polinómio interpolador $P_n(x)$ no intervalo $[a,b]$ e ao desvio definido por, $P_n(x) - f(x)$, dá-se o nome de erro de interpolação.

Teorema 1

Sejam $x_0, x_1, ..., x_n$, $n+1$ nós distintos num intervalo $[a,b]$. Então, para os valores arbitrários $y_0, y_1, ..., y_n$, existe um único polinómio P_n de grau menor ou igual a n, tal que

$$P_n(x_i) = y_i = f(x_i), \qquad 0 \le i \le n \tag{2.2}$$

Demonstração:

Primeiro provaremos a unicidade do polinómio interpolador. Para isso, suponhamos que existem dois polinómios interpoladores dos valores $y_0, y_1, ..., y_n$ nos nós distintos $x_0, x_1, ..., x_n$, P_n e Q_n

Desta forma, $P_n - Q_n$, a diferença entre os dois polinómios de grau menor ou igual a n, será também um polinómio de grau menor ou igual a n, com a seguinte propriedade $(P_n - Q_n)(x_i) = 0$ para $0 \le i \le n$, visto que os dois polinómios assumem os mesmos valores nodais.

Como os nós são distintos, o polinómio $P_n - Q_n$ de grau menor ou igual a n, possuirá $n+1$ zeros, pelo que terá de ser um polinómio nulo, ou seja $P_n - Q_n = 0$, donde concluímos que $P_n \equiv Q_n$.

Quanto à existência do polinómio interpolador:

sabendo que um polinómio interpolador de grau menor ou igual a n, pode ser escrito na forma, $p_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$, pela definição tem-se $P_n(x_i) = y_i = f(x_i)$, $0 \le i \le n$. Daqui, resulta que os coeficientes do polinómio podem ser determinados resolvendo o seguinte sistema linear

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ & & & & \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \ldots \\ a_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \ldots \\ f(x_n) \end{bmatrix}$$

onde $V(x_0, x_1, \ldots, x_n) = \begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ & & & & \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix}$ é a matriz do sistema, conhecida por matriz de Vandermonde (Mirsky, 1955).

O determinante da matriz de Vandermonde é dado por, $\det(V(x_0, x_1, \ldots, x_n)) = \prod_{\substack{i,j=0 \\ j>i}}^{n} (x_j - x_i)$ (Pina, 1995), que, neste caso, é diferente de zero, pois, os nós de interpolação são distintos.

Por conseguinte, o sistema tem solução única, ou seja, o polinómio interpolador existe e é único.

2.1. Polinómio interpolador de Lagrange

Consideremos a seguinte tabela de valores, para uma função continuamente diferencial até a ordem $n+1$.

x	x_0	x_1	...	x_n
$f(x)$	$f(x_0)$	$f(x_1)$	...	$f(x_n)$

Para estes dados, é possível construir um único polinómio interpolador, de grau menor ou igual a n.

Visto que este polinómio terá que interpolar todos os valores nodais, $f(x_0), f(x_1),..., f(x_n)$, poderá ser escrito como sendo uma combinação linear das ordenadas, ou seja

$$P_n(x) = l_0(x)f(x_0) + l_1(x)f(x_1) + ... + l_n(x)f(x_n) = \sum_{i=0}^{n} l_i(x)f(x_i) \tag{2.3}$$

Onde $l_i(x), \quad i = 0,1,...,n$ são polinómios de grau n. O polinómio $P_n(x)$, por ser um polinómio interpolador, deve obrigatoriamente satisfazer a condição (2.1), que é o mesmo que dizer o seguinte

$$\begin{cases} f(x_0) = P_n(x_0) = l_0(x_0)f(x_0) + l_1(x_0)f(x_1) + ... + l_n(x_0)f(x_n) \\ f(x_1) = P_n(x_1) = l_0(x_1)f(x_0) + l_1(x_1)f(x_1) + ... + l_n(x_1)f(x_n) \\ \vdots \\ f(x_n) = P_n(x_n) = l_0(x_n)f(x_0) + l_1(x_n)f(x_1) + ... + l_n(x_n)f(x_n) \end{cases} \tag{2.4}$$

O sistema de equações (2.4), só será possível, se os polinómios $l_i(x)$ satisfazerem as seguintes condições

$$l_i(x_j)=\begin{cases}1, & i=j\\ 0, & i\neq j\end{cases} \tag{2.5}$$

Tendo em conta que $l_i(x)=0$ em $x=x_0,x_1,...,x_{i-1},x_{i+1},...x_n$, podemos concluir que $(x-x_0)$, $(x-x_1)$, …, $(x-x_{i-1})$, $(x-x_{i+1})$, …, $(x-x_n)$ são factores de $l_i(x)$. O produto desses factores é um polinómio de grau n.

O polinómio $l_i(x)$, será então dado por

$$l_i(x)=K(x-x_0)(x-x_1)...(x-x_{i-1})(x-x_{i+1})...(x-x_n).$$

Onde K é uma constante, que será determinada de modo a que o polinómio $l_i(x)$ satisfaça a condição (2.5), ou seja, $l_i(x_i)=1$.

Obtém-se então

$$l_i(x_i)=1=K(x_i-x_0)(x_i-x_1)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n),$$

ou seja

$$K=\frac{1}{(x_i-x_0)(x_i-x_1)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)},$$

donde os polinómios, $l_i(x), \quad i=0,1,...,n$ serão obtidos da seguinte forma

$$l_i(x)=\frac{(x-x_0)(x-x_1)...(x-x_{i-1})(x-x_{i+1})...(x-x_n)}{(x_i-x_0)(x_i-x_1)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)}. \tag{2.6}$$

Definição 3. O polinómio definido pela expressão (2.3), com $l_i(x)$ definidos tal como em (2.6), é designado por polinómio interpolador de Lagrange (Iyengar, 2009) e os $l_i(x)$ são chamados de polinómios fundamentais de Lagrange.

Os polinómios fundamentais de Lagrange, também podem ser escritos, de uma forma mais compacta.

Para o efeito, consideremos o polinómio nodal[2] $W_n(x)$, definido por

$$W_n(x) = (x - x_0)(x - x_1)...(x - x_n) \tag{2.7}$$

Derivando o polinómio nodal, $W_n(x)$ em ordem a x e considerando $x = x_i$, obtém-se

$$W_n'(x_i) = (x_i - x_0)(x_i - x_1)...(x_i - x_{i-1})(x_i - x_{i+1})...(x_i - x_n).$$

Deste modo, podemos escrever os polinómios fundamentais de Lagrange na forma

$$l_i(x) = \frac{W_n(x)}{(x - x_i)W_n'(x_i)} \tag{2.8}$$

Exemplo 1

Utilizando o polinómio interpolador de Lagrange, de grau ≤ 3, que interpola os valores $y_i = f(x_i)$ representados no quadro que se segue, determinar uma estimativa para $f(2.5)$:

i	0	1	2	3
x_i	0	2	3	4
y_i	2	1	-1	4

[2] Polinómio que se anula em todos os nós de interpolação.

Pela expressão (2.6), calculam-se os polinómios fundamentais de Lagrange, $l_0(x)$, $l_1(x)$, $l_2(x)$ e $l_3(x)$, associados aos quatro nós de interpolação.

$$l_0(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} = \frac{(x-2)(x-3)(x-4)}{(0-2)(0-3)(0-4)} = -\frac{1}{24}(x-2)(x-3)(x-4)$$

$$l_1(x) = \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} = \frac{(x-0)(x-3)(x-4)}{(2-0)(2-3)(2-4)} = \frac{1}{4}x(x-3)(x-4)$$

$$l_2(x) = \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} = \frac{(x-0)(x-2)(x-4)}{(3-0)(3-2)(3-4)} = -\frac{1}{3}x(x-2)(x-4)$$

$$l_3(x) = \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} = \frac{(x-0)(x-2)(x-3)}{(4-0)(4-2)(4-3)} = \frac{1}{8}x(x-2)(x-3)$$

O polinómio interpolador na fórmula de Lagrange será então dado pela expressão (2.3), ou seja,

$$P_3(x) = \sum_{i=0}^{3} l_i(x) y_i$$

$$= -\frac{1}{12}(x-2)(x-3)(x-4) + \frac{1}{4}x(x-3)(x-4) + \frac{1}{3}x(x-2)(x-4) + \frac{1}{2}x(x-2)(x-3)$$

Deste modo, $f(2.5) \approx P_3(2.5) = -0.5$.

Este resultado poderia ser obtido utilizando a rotina **lagrange**, desenvolvida em Matlab, da seguinte forma:

```
function p = lagrange(t,x,y)

%Entrada - t é um vector que indica a(s) abcissa(s) onde
%         queremos calcular o(s) valor(es) de Pn(x)
%        - x é um vector que contém os nós
%        - y é um vector que contém os valores nodais
%Saída   - p é um vector que contém o(s) valor(es) de Pn(x)
%         calculado(s) na(s) abcissa(s) de t
```

```
c = lagrangecoef(x,y);
m = length(x);
for i = 1 : length(t)
  p(i) = 0;
  for j = 1 : m
    N(j) = 1;
    for k = 1 : m
      if j ~= k
        N(j) = N(j) * (t(i) - x(k));
      end
    end
    p(i) = p(i) + N(j) * c(j)*y(j);
  end
end
```

que por sua vez precisará da rotina também desenvolvida em MatLab, **lagrangecoef**.

```
function c = lagrangecoef(x,y)

%Entrada - x é um vector que contém os nós
%       - y é um vector que contém os valores nodais
%Saída   - c é um vector que contém os coeficientes dos
%         polinómios de Lagrange associados aos nós

n=length(x);
for k = 1 : n
  d(k) = 1;
  for i = 1 : n
    if i ~= k
      d(k)=d(k)*(x(k) - x(i));
    end
    c(k) = 1/d(k);
  end
end
```

Para o efeito, no ambiente de trabalho do matlab, chamamos a função **lagrange**, indicando o ponto onde se pretende calcular o valor da função, os nós de interpolação e os respectivos valores nodais.

```
>> lagrange(2.5,[0,2,3,4],[2,1,-1,4])
ans =
  -0.5000
```

Apesar da sua simplicidade, o polinómio interpolador na fórmula de Lagrange apresenta algumas desvantagens, como por exemplo:

- Requer um número de operações elevado;
- Os polinómios estão associados a um conjunto de nós, pelo que uma mudança da posição e/ou do número destes, altera por completo o polinómio, havendo necessidade de refazer todos os cálculos dos polinómios de Lagrange.

2.2. Polinómio interpolador de Newton nas diferenças divididas

Consideremos conhecidos os pontos $(x_i, f(x_i)), \quad i = 0,1,...,n$.

Para quaisquer dois pontos consecutivos, $(x_i, f(x_i))$ e $(x_{i+1}, f(x_{i+1}))$, as diferenças divididas de primeira ordem são definidas por

$$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}, \qquad i = 0,1,...,n-1 \tag{2.9}$$

Da mesma forma, considerando quaisquer três pontos consecutivos, $(x_i, f(x_i))$, $(x_{i+1}, f(x_{i+1}))$ e $(x_{i+2}, f(x_{i+2}))$, as diferenças divididas de segunda ordem serão definidas por

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}, \qquad i = 0,1,...,n-2 \tag{2.10}$$

De uma forma geral, considerando quaisquer $n+1$ pontos consecutivos, a diferença dividida de ordem n, usando todos os $n+1$ pontos, será definida por

$$f[x_0, x_1, ..., x_n] = \frac{f[x_1, ..., x_n] - f[x_0, ...x_{n-1}]}{x_n - x_0} \tag{2.11}$$

Resumindo, as diferenças divididas são definidas recursivamente do seguinte modo

$$
\begin{aligned}
&f[x_k] = f(x_k) \\
&f[x_{k-1}, x_k] = \frac{f[x_k] - f[x_{k-1}]}{x_k - x_{k-1}} \\
&\ldots \\
&f[x_{k-j}, x_{k-j+1}, \ldots, x_k] = \frac{f[x_{k-j+1}, \ldots x_k] - f[x_{k-j}, \ldots, x_{k-1}]}{x_k - x_{k-j}}
\end{aligned}
\tag{2.12}
$$

Para uma melhor compreensão, as diferenças divididas podem ser escritas numa tabela adequada como a que se mostra a seguir para o caso de cinco pontos.

x	$f(x)$	1ª ordem	2ª ordem	3ª ordem	4ª ordem
x_0	$f(x_0)$				
		$f[x_0, x_1]$			
x_1	$f(x_1)$		$f[x_0, x_1, x_2]$		
		$f[x_1, x_2]$		$f[x_0, x_1, x_2, x_3]$	
x_2	$f(x_2)$		$f[x_1, x_2, x_3]$		$f[x_0, x_1, x_2, x_3, x_4]$
		$f[x_2, x_3]$		$f[x_1, x_2, x_3, x_4]$	
x_3	$f(x_3)$		$f[x_2, x_3, x_4]$		
		$f[x_3, x_4]$			
x_4	$f(x_4)$				

Tabela 1 - Diferenças divididas de 4ª ordem

Como forma de evitar os inconvenientes da fórmula de Lagrange, usa-se o polinómio escrito na forma de Newton.

Deste modo, tomando os nós $x_0, x_1, \ldots, x_{n-1}$ como centros do polinómio, este pode ser escrito na forma

$$
P_n(x) = a_0 + a_1 W_0(x) + a_2 W_1(x) + \ldots + a_n W_{n-1}(x) \tag{2.13}
$$

onde os coeficientes do polinómio são determinados por forma a que P_n seja de facto polinómio interpolador dos valores $y_0, y_1, ..., y_n$, nos nós distintos $x_0, x_1, ..., x_n$.

Caso seja necessário construir o polinómio P_{n+1} que interpola nos nós $x_0, x_1, ..., x_n, x_{n+1}$ os valores nodais $y_0, y_1, ..., y_n, y_{n+1}$, basta acrescentar ao polinómio P_n um termo da forma $a_{n+1}W_n(x)$, ou seja,

$$P_{n+1}(x) = P_n(x) + a_{n+1}W_n(x) \ . \tag{2.14}$$

Cada coeficiente a_k do polinómio $P_n(x)$ depende dos valores $f(x_j)$, para $j = 0,1,...,k$.

Teorema 2 (Polinómio de Newton)

Consideremos $x_0, x_1, ..., x_n$, $n+1$ nós distintos, num intervalo $I = [a,b]$.

Existe um único polinómio $P_n(x)$ de grau menor ou igual a n, satisfazendo as condições

$$f(x_j) = P_n(x_j), \text{ para } j = 0,1,...,n,$$

que pode ser escrito na forma de Newton (Pina, 1995), ou seja

$$P_n(x) = a_0 + a_1(x - x_0) + ... + a_n(x - x_0)(x - x_1)...(x - x_{n-1}), \tag{2.15}$$

onde $a_k = f[x_0, x_1, ...x_k], \qquad k = 0,1,...,n.$ (2.16)

Exemplo 2

Para uma função $y = f(x)$ conhecida apenas num número finito de pontos, calcular uma aproximação para $f(2.5)$, usando para o efeito, um polinómio interpolador de grau ≤ 3, que interpola os valores representados no quadro que se segue, na fórmula de Newton, usando diferenças divididas.

i	0	1	2	3
x_i	0	2	3	4
y_i	2	1	-1	4

O polinómio interpolador será obtido pela expressão (2.15), onde os coeficientes a_i, $i = 0,...,3$ serão calculados pela expressão (2.16), ou seja, $a_i = f[x_0, x_1, ...x_k]$, $k = 0,1,...,i$, que são obtidos por diferenças divididas.

Para este caso, é possível construir a seguinte tabela de diferenças divididas:

x	$f(x)$	1ª ordem	2ª ordem	3ª ordem
$x_0 = 0$	2			
		$\frac{1-2}{2-0} = -\frac{1}{2}$		
$x_1 = 2$	1		$\frac{-2-\left(-1/2\right)}{3-0} = -\frac{1}{2}$	
		$\frac{-1-1}{3-2} = -2$		$\frac{7/2-\left(-1/2\right)}{4-0} = 1$
$x_2 = 3$	-1		$\frac{5-(-2)}{4-2} = \frac{7}{2}$	
		$\frac{4-(-1)}{4-3} = 5$		
$x_3 = 4$	4			

Tabela 2 - Diferenças divididas correspondentes aos dados do exemplo **2**

O polinómio interpolador de Newton de grau três será, então, dado por

$$P_3(x) = 2 - \frac{1}{2}x - \frac{1}{2}x(x-2) + x(x-2)(x-3)$$

Pelo que, $f(2.5) \approx P_3(2.5) = -0.5$.

Esse resultado poderia ser obtido, usando a rotina, desenvolvida em MatLab, **newtondifdiv**, que tem como entrada os vectores, t, x e y, conforme se pode ver a seguir.

```
% Calcula o(s) valor(es) do Polinómio interpolador
% de Newton Pn, num ou mais pontos, usando as diferenças divididas

 function p = newtondifdiv(t,x,y)

% Entrada - t é um vector que indica a(s) abcissa(s) onde
%          queremos calcular o(s) valor(es) de Pn(x)
%         - x é um vector que contém os nós
%             - y é um vector que contém os valores nodais
% Saída   - p é um vector que contém o(s) valor(es) de Pn(x)
%          calculado(s) na(s) abcissa(s) de t
a = difdivcoef(x,y);
n = length(x);
for i = 1 : length(t)
   d(1) = 1;
   c(1) = a(1);
   for j = 2 : n
      d(j) = (t(i) - x(j-1)).*d(j-1);
      c(j) = a(j).*d(j);
   end
   p(i) = sum(c);
end
```

>> **newtondifdiv**(2.5,[0,2,3,4],[2,1,-1,4])

Tabela das diferenças divididas

```
   -0.5000   -0.5000    1.0000

   -2.0000    3.5000         0

    5.0000         0         0
```

ans =

```
   -0.5000.
```

Esta utiliza ainda como auxiliar a seguinte rotina **difdivcoef**.

```
% Calcula as diferenças divididas que figuram no
% polinómio interpolador de Newton nas diferenças
% divididas, Pn(x):
% a=[a0 a1 ... an], com Pn(x)=a0+a1(x-x0)+a2(x-x0)(x-x1)+...
%                    +an(x-x0)(x-x1)...(x-xn-1)
%
% d é uma matriz auxiliar que sob a forma diagonal superior,
```

```
% nos fornece a tabela de diferenças divididas (diferenças
% de primeira ordem em diante)

function a = difdivcoef(x,y)

%Entrada - x é um vector que contém os nós
%        - y é um vector que contém os valores nodais
%Saída   - a é um vector que contém as diferenças divididas
%          que figuram no polinómio interpolador de Newton

n = length(x);
a(1) = y(1);

% cálculo das diferenças divididas de primeira ordem

for k = 1 : n-1
  d(k,1) = (y(k+1) - y(k))/(x(k+1) - x(k));
end

% cálculo das diferenças divididas de ordem j

for j = 2 : n-1
  for k = 1 : n-j
    d(k,j) = (d(k+1,j-1) - d(k,j-1))/(x(k+j) - x(k));
  end
end

disp('Tabela das diferenças divididas')
disp(d)

for j = 2 : n
  a(j) = d(1,j-1);
end
```

2.3. Interpolação inversa

Consideremos os nós $x_0, x_1, ..., x_n$ pertencentes ao intervalo $[a,b]$ e, $y_0, y_1, ..., y_n$, os valores nodais de uma função f, diferenciável em $[a,b]$.

Se a função f for estritamente monótona (crescente ou decrescente), possuirá inversa, pelo que se torna possível escrever

$$x = g(y) \tag{2.17}$$

Onde g é a função inversa de f.

Deste modo, será possível construir um polinómio $P(y)$ que interpola os valores nodais $x_0, x_1, ..., x_n$, nos nós $y_0, y_1, ..., y_n$. A este processo dá-se o nome de interpolação inversa.

Este resultado é frequentemente utilizado em problemas do tipo $f(x) = c$, onde se pretende calcular o valor de x.

Para o efeito, tendo em conta a função inversa de f, g (pressupõe-se que f é estritamente monótona), o valor de x será dado por $x = g(c) \approx P(c)$, onde P é um polinómio interpolador em y.
Uma das aplicações mais comuns é a determinação de zeros de funções, como a do exemplo seguinte.

Exemplo 3

Determinar aproximadamente o zero da função $f(x) = x^2 - \mathbf{exp}(-x) + \mathbf{ln}(x+1)$ no intervalo $[0,1]$.

Como é fácil verificar, $f'(x) = 2x + \mathbf{exp}(-x) + \frac{1}{x+1}$, sendo esta derivada sempre positiva no intervalo $[0,1]$, o que implica que a função f é monótona crescente nesse intervalo, pelo que, possui uma inversa, g.

Deste modo, o problema $f(x) = 0$ poderá ser resolvido da seguinte forma $x = g(0)$.

Vamos então construir o polinómio P que interpola a função g, considerando os pontos representados na tabela que se segue

x	0	0.25	0.5	0.75	1
$y = f(x)$	-1	-0.4931	0.0489	0.6497	1.3252

Pela tabela de diferenças divididas para a função g, na qual $y = f(x)$ passam a ser os nós e x os valores nodais, chega-se aos seguintes coeficientes

$$g(y_0) = 0$$
$$g[y_0, y_1] = 0.4932$$
$$g[y_0, y_1, y_2] = -0.0305$$
$$g[y_0, y_1, y_2, y_3] = -0.0055$$
$$g[y_0, y_1, y_2, y_3, y_4] = 0.0032$$

O polinómio interpolador de grau quatro será dado por

$$P(y) = 0.4932(y+1) - 0.0305(y+1)(y+0.4931) - 0.0055(y+1)(y+0.4931)(y-0.0489) + \\ = +0.0032(y+1)(y+0.4931)(y-0.0489)(y-0.6497)$$

Sendo assim, a solução aproximada do problema $f(x) = 0$ será

$$x \approx P(0) = 0.4783,$$

sendo este valor preciso até à quarta casa decimal, uma vez que o valor "exacto" é dado por $x = 0.4783765890$.

2.4. Erro de interpolação

2.4.1. Análise de erros

O polinómio interpolador coincidente com f nos $n+1$ nós de interpolação, $x_0, x_1, ..., x_n$ pode ser usado para aproximar a função f em qualquer ponto do intervalo $[a,b]$, contendo este os nós de interpolação. Deste modo, torna-se imperativo estudar o erro cometido na interpolação, ou seja, saber o quão longe ou perto o polinómio interpolador está da função interpolada num dado ponto de $[a,b]$.

Teorema 3

Sejam $f \in C^k[a,b]$ e $x_0, x_1, ..., x_k$, $k+1$ nós distintos pertencentes ao intervalo $[a,b]$. Então, existe um ponto $\xi \in (a,b)$, tal que

$$f[x_0, x_1, ..., x_k] = \frac{1}{k!} f^{(k)}(\xi) \qquad (2.18)$$

Demonstração

Para $k=1$ o teorema reduz-se ao teorema do valor médio, pelo que é obviamente verdadeiro.

Para o caso geral, consideremos a expressão

$$e_k(x)=f(x)-P_k(x) \tag{2.19}$$

onde P_k é o polinómio de grau menor ou igual a k que interpola f nos nós distintos $x_0,x_1,...,x_k$.

A função $e_k(x)$, dada pela expressão (2.19) terá então no mínimo $k+1$ zeros no intervalo $[a,b]$, pelo que $e'_k(x)=f'(x)-P'_k(x)$ terá no mínimo k zeros no mesmo intervalo.

Seguindo o mesmo raciocínio, a função $e_k^{(k)}(x)=f^{(k)}(x)-P_k^{(k)}(x)$ terá pelo menos um zero no intervalo $[a,b]$.

Seja ξ esse zero, deste modo

$$e_k^{(k)}(\xi)=f^{(k)}(\xi)-P_k^{(k)}(\xi)=0$$

donde se concluí que

$$P_k^{(k)}(\xi)=f^{(k)}(\xi)=k!\,f[x_0,x_1,...,x_k] \qquad \blacksquare$$

Obs.

$$P_k(x)=f[x_0]+f[x_0,x_1](x-x_0)+...+f[x_0,x_1,...x_k](x-x_0)(x-x_1)...(x-x_{k-1})$$

Sabendo que de cada vez que se deriva um polinómio, este diminui o seu grau em uma unidade, na derivada de ordem k, o grau será 0 e o coeficiente dado por

$$k\times(k-1)\times(k-2)\times...\times 2\times 1\,f[x_0,x_1,...,x_k]=k!\,f[x_0,x_1,...,x_k]$$

Teorema 4

Seja a função f, tal que $f\in C^{n+1}[a,b]$ e, seja P_n o polinómio de grau menor ou igual a n, que interpola a função f nos $n+1$ pontos distintos, $x_0,x_1,...,x_n$ pertencentes ao intervalo $[a,b]$. Para cada $x\in[a,b]$, existirá um ponto $\xi\in(a,b)$ tal que

$$e_n(x)=f(x)-P(x)=\frac{1}{(n+1)!}f^{(n+1)}(\xi)\prod_{i=0}^{n}(x-x_i) \tag{2.20}$$

Demonstração

Se x for um dos nós de interpolação, a prova é óbvia, já que ambos os membros da igualdade se reduzem a zero.

Sendo assim, seja $x = x^*$ um ponto qualquer do intervalo $[a,b]$, que não seja coincidente com um nó de interpolação. O polinómio P_{n+1} que interpola a função f nos pontos $x_0, x_1, ..., x_n, x^*$ será dado na forma de Newton por

$$P_{n+1}(x) = P_n(x) + f[x_0, x_1, ..., x_n, x^*]W_n(x)$$

Donde resulta que

$$f(x^*) = P_{n+1}(x^*) = P_n(x^*) + f[x_0, x_1, ..., x_n, x^*]W_n(x^*).$$

Substituindo na expressão do erro, vem que

$$e(x^*) = f(x^*) - P_n(x^*) = f[x_0, x_1, ..., x_n, x^*]W_n(x^*)$$

Recorrendo ao teorema 3 e, substituindo a variável x^* por x, determina-se o coeficiente $f[x_0, x_1, ..., x_n, x^*]$, que será dado por

$f[x_0, x_1, ..., x_n, x^*] = \frac{1}{(k+1)!} f^{(k+1)}(\xi)$, chegando-se deste modo ao resultado pretendido.

■

No entanto, tendo em conta que o valor de ξ é desconhecido, na prática apenas é possível estimar o valor do erro de interpolação dado pela expressão (2.20).

Deste modo, recorre-se ao majorante do erro, que será dado por

$$|e_n(x)| = \frac{1}{(n+1)!} \left| f^{(n+1)}(\xi) \right| \left| \prod_{i=0}^{n} (x - x_i) \right|$$

$$\leq \frac{1}{(n+1)!}\left[\max_{a\leq x\leq b}\left|f^{(n+1)}(x)\right|\right]W_n(x)| \qquad (2.21)$$

Exemplo 4.

Qual deve ser o espaçamento mínimo entre nós consecutivos, de modo a que na interpolação linear da função $f(x)=e^x-1$, no intervalo $I=[0,1]$, o erro não exceda em valor absoluto 0.1×10^{-4}?

Tendo em conta a expressão (2.21), o erro de interpolação linear é majorado por

$$|e_1|\leq\frac{1}{2}\left[\max_{x\in I}\left|f^{''}(x)\right|\right]\left[\max_{x\in I}|W_1(x)|\right]$$

com $W_1(x)=(x-x_0)(x-x_1)$. Derivando a polinómio nodal $W_1(x)$, obtém-se que $W_1^{'}(x)=2x-x_0-x_1$, que se anula para $x=\frac{x_0+x_1}{2}$, pelo que $|W_1(x)|$ possuirá um máximo nesse ponto.

Atendendo a que o espaçamento é dado por $h=x_1-x_0$, o valor máximo em valor absoluto do polinómio nodal será dado por $\frac{h^2}{4}$, pelo que

$$|e_1|\leq\frac{h^2}{8}\left[\max_{x\in I}\left|f^{''}(x)\right|\right]$$

Para o caso em estudo, a segunda derivada da função é dado por $f^{''}(x)=e^x>0$. Tendo em conta que a terceira derivada é $f^{'''}(x)=e^x$ que é sempre positiva no intervalo $I=[0,1]$, conclui-se que o valor máximo da segunda derivada ocorre no limite superior do intervalo, ou seja, $\max_{x\in[0,2]}\left|f^{''}(x)\right|=e^1$.

O objectivo inicial poderá ser garantido, resolvendo o seguinte problema, $\frac{h^2}{8}e^1 \leq 0.1\times10^{-4}$, ou seja, $h \leq 0.542\times10^{-2}$.

Exemplo 5

Tendo em conta os dados do exemplo 2, obter uma estimativa do erro cometido no cálculo do valor aproximado para $f(1.5)$ quando se utiliza um polinómio interpolador de grau 1.

Considerando os nós de interpolação, $x_0 = 0$ e $x_1 = 2$, obtém-se o polinómio interpolador $P_1(x) = 2 - \frac{1}{2}x$, pelo que, $f(1.5) \approx P_1(1.5) = 1.25$.

Pelo teorema 4, o erro cometido na interpolação linear (grau 1) é majorado por

$$|e_1(1.5)| \leq \frac{1}{2!}\left[\max_{a\leq x\leq b}|f''(x)|\right]|W_1(1.5)|$$

Tendo em conta o teorema 3, o valor máximo da derivada de segunda ordem da função pode ser estimado por

$$\max|f''(x)| \approx 2!\times\max|f[x_0, x_1, x_2]| = 2\times\frac{7}{2} = 7.$$

O erro de interpolação cometido será, então, majorado por

$$|e_1(1.5)| \leq \frac{1}{2!}\times 7\times|(1.5-0)(1.5-2)| = 2.625.$$

Note-se que, neste caso, a função interpolada não é conhecida e esta estimativa grosseira obtida para o erro de interpolação pode ser explicada por dois motivos. Por um lado, pela estimativa da derivada de segunda ordem da função, por outro, pelo valor do polinómio nodal, $W_1(1.5) = 0.75$, o que denota a influência que a distância entre os nós de interpolação pode assumir na qualidade da aproximação obtida quando se interpola uma dada função.

2.4.2. Rigidez dos polinómios interpoladores

A interpolação polinomial, como já foi referido, consiste em aproximar uma determinada função, por um polinómio. Deste modo, o que se pretende é que esse polinómio interpolador esteja "próximo" da função interpolada.

Uma questão que de imediato pode ser levantada é: será que quanto maior for o grau do polinómio interpolador, melhor será a aproximação obtida, ou seja, será que o erro de interpolação tende para zero, quando o grau n desse polinómio tende para infinito?

A resposta é que nem sempre tal sucede. Note-se que, a este respeito o teorema de Weierstrass (1885) garante que uma função contínua pode ser aproximada por polinómios com a precisão que se queira, no entanto, este teorema não afirma que estes polinómios possam ser obtidos por interpolação.

Como se pode verificar de seguida, com um exemplo célebre, descoberto por Runge (1901), não se pode garantir que tal aconteça.

Para o efeito, considera-se a função de Runge, definida por

$$f(x)=\frac{1}{1+25x^2}, \qquad x\in[-1,1]$$

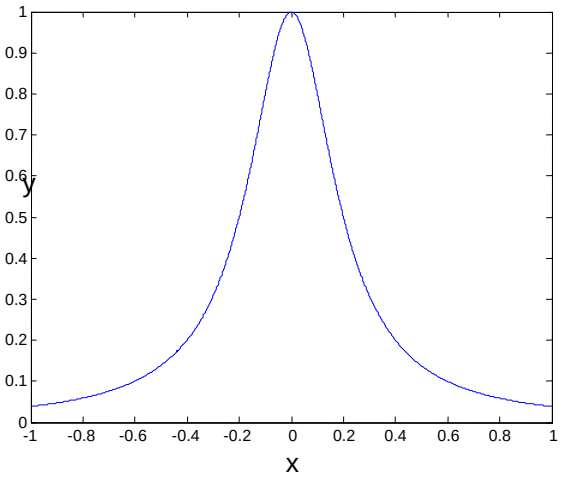


Figura 1 – Função de Runge para x $\in$ [-1,1]

que se pretende interpolar por um polinómio de um dado grau n, usando nós equidistantes, pertencentes ao intervalo $[-1,1]$.

Como pode ser observado na figura 2, à medida que n aumenta, com $n=4$ e $n=9$, o polinómio interpolador P_n desenvolve oscilações bastantes acentuadas, pelo que o erro de interpolação e_n não tende para zero e, consequentemente, o polinómio P_n não converge para a função interpolada f.

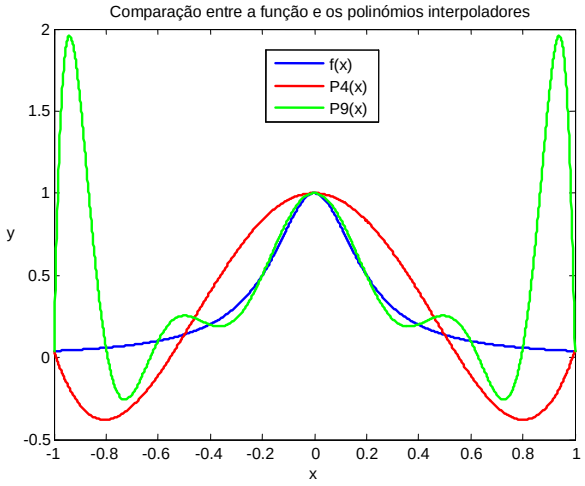


Figura 2 - Interpolação da função de Runge com número de nós diferentes.

As oscilações verificadas estão intimamente ligadas ao facto de que, para polinómios de grau elevado, $P_n(x)=a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$, os coeficientes das suas derivadas, $na_n, (n-1)a_{n-1}, ..., a_1$, são, em geral, maiores, em valor absoluto, que os coeficientes de P_n, tornando assim possíveis variações e oscilações importantes, nomeadamente, próximo dos extremos do intervalo de interpolação.

Atendendo à expressão do erro de interpolação, conclui-se que, para que a interpolação polinomial produza polinómios que aproximem tão bem quanto se queira a função interpolada, é necessário que esta satisfaça certas propriedades de regularidade no que diz respeito à sua derivada de ordem $n+1$ e que a localização dos nós de interpolação, seja tal que, o valor absoluto do polinómio nodal W_n se torne mínimo.

Resumindo, a função f, bem como os nós de interpolação devem ser, tal que, os valores absolutos da derivada de ordem $n+1$, bem como o valor absoluto do polinómio nodal, variem com n de tal forma que o segundo membro da expressão (2.20) tenda para zero quando n tende para infinito.

Para contornar os inconvenientes apontados à interpolação da função de Runge, onde ocorrem oscilações acentuadas, abordaremos duas técnicas, uma delas na secção seguinte, usando os designados nós de Chebyshev, e a outra com recurso a interpolação com Splines, que será tema do capítulo 3.

Como a seguir veremos, fazendo coincidir os nós de interpolação com os zeros dos chamados polinómios de Chebyshev (Mason, 2003), torna-se possível minimizar o polinómio nodal W_n, em valor absoluto, e, consequentemente, o erro de interpolação.

2.4.3. Nós de Chebyshev

Como se verifica na expressão (2.21), o erro de interpolação depende da derivada de ordem $n+1$ da função interpolada e do polinómio nodal. Como é possível ver na figura 3, os valores máximos do polinómio nodal W_n localizam-se na proximidade dos extremos do intervalo de interpolação e a sua magnitude está relacionada com o número de nós. Verifica-se que a magnitude daqueles máximos diminui à medida que o número de nós aumenta.

Para o efeito, definimos os polinómios nodais, $W_5(x)$, $W_6(x)$ e $W_7(x)$ com nós equidistantes no intervalo $[-1,1]$, ou seja,

$$W_5(x) = (x+1)\left(x+\frac{3}{5}\right)\left(x+\frac{1}{5}\right)\left(x-\frac{1}{5}\right)\left(x-\frac{3}{5}\right)(x-1)$$

$$W_6(x) = (x+1)\left(x+\frac{2}{3}\right)\left(x+\frac{1}{3}\right)x\left(x-\frac{1}{3}\right)\left(x-\frac{2}{3}\right)(x-1)$$

$$W_7(x) = (x+1)\left(x+\frac{5}{7}\right)\left(x+\frac{3}{7}\right)\left(x+\frac{1}{7}\right)\left(x-\frac{1}{7}\right)\left(x-\frac{3}{7}\right)\left(x-\frac{5}{7}\right)(x-1)$$

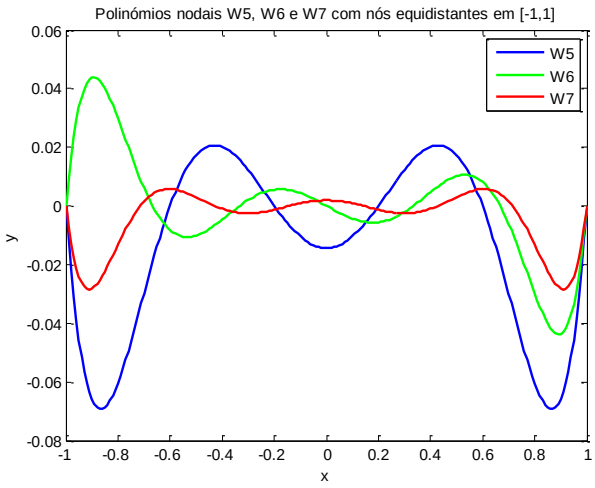


Figura 3 - Polinómios nodais, com número de nós diferentes

Deste modo, é de esperar que os polinómios interpoladores produzam bons resultados, quando utilizados na zona central do intervalo de interpolação e piorem nos extremos desse intervalo.

Torna-se, então, importante, saber qual a posição dos nós que conduzem a valores mínimos do polinómio nodal. Tal localização é conseguida quando os nós de interpolação coincidem com os zeros dos polinómios de Chebyshev.

Para o efeito, supõe-se que o intervalo de interpolação é $[-1,1]$. Se tal não acontecer, utiliza-se a seguinte relação que permite transformar o intervalo $[-1,1]$ em qualquer intervalo $[a,b]$

$$x = a\frac{1-\xi}{2} + b\frac{1+\xi}{2}, \qquad \xi \in [-1,1]\,. \tag{2.22}$$

Definição 4

A função definida por $T_n(x) = \mathbf{cos}(n\,\mathbf{arccos}(x))$ é um polinómio de grau n, conhecido por polinómio de Chebyshev (Pina, 1995).

Para $n=0$ e $n=1$, deduz-se imediatamente que

$$T_0(x)=1 \text{ e } T_1(x)=x\,.$$

Tendo em conta a seguinte mudança de variável, $x=\mathbf{cos}(\theta)$, obtém-se que $T_n(\mathbf{cos}(\theta))=\mathbf{cos}(n\theta)$.

Usando a fórmula do co-seno de uma soma ($\mathbf{cos}(\alpha+\beta)=\mathbf{cos}(\alpha)\mathbf{cos}(\beta)-\mathbf{sin}(\alpha)\mathbf{sin}(\beta)$), torna-se possível escrever que

$$T_{n+1}(\mathbf{cos}(\theta))=\mathbf{cos}((n+1)\theta)=\mathbf{cos}(n\theta)\mathbf{cos}(\theta)-\mathbf{sin}(n\theta)\mathbf{sin}(\theta)$$

$$T_{n-1}(\mathbf{cos}(\theta))=\mathbf{cos}((n-1)\theta)=\mathbf{cos}(n\theta)\mathbf{cos}(\theta)+\mathbf{sin}(n\theta)\mathbf{sin}(\theta)$$

Somando ambos os membros das duas expressões e, após alguns arranjos, obtém-se a seguinte expressão

$$\mathbf{cos}((n+1)\theta)=2\mathbf{cos}(n\theta)\mathbf{cos}(\theta)-\mathbf{cos}((n-1)\theta).$$

Tendo em conta que $x=\mathbf{cos}(\theta)$ (logo, $\theta=\mathbf{arccos}(x)$), tem-se

$$\mathbf{cos}((n+1)\mathbf{arccos}(x))=2\mathbf{cos}(n\mathbf{arccos}(x))x-\mathbf{cos}((n-1)\mathbf{arccos}(x))\ ,$$

donde, daqui pode obter-se a seguinte fórmula de recorrência

$$T_{n+1}(x)=2xT_n(x)-T_{n-1}(x)\ .$$

Aplicando sucessivamente esta fórmula de recorrência, tem-se que

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_2(x) = 2x^2 - 1$$
$$T_3(x) = 4x^3 - 3x$$
$$T_4(x) = 8x^4 - 8x^2 + 1$$
$$T_5(x) = 16x^5 - 20x^3 + 5x$$
...

Teorema 5

O polinómio de Chebyshev $T_n(x) = \mathbf{cos}(n\,\mathbf{arccos}(x))$ tem os zeros localizados nos pontos

$$x_k = \mathbf{cos}\frac{(2k-1)\pi}{2n}, \qquad k = 1,...,n \tag{2.23}$$

e os extremos localizados nos pontos

$$x_k^{'} = \mathbf{cos}\frac{k\pi}{n}, \qquad k = 0,1,...,n \tag{2.24}$$

nos quais

$$T_n\left(x_k^{'}\right) = (-1)^k . \tag{2.25}$$

Demonstração:

Como pode ser verificado,

$$T_n(x_k) = \mathbf{cos}\left(n\,\mathbf{arccos}\left(\mathbf{cos}\left(\frac{2k-1}{2n}\pi\right)\right)\right) = \mathbf{cos}\left(\frac{2k-1}{2}\pi\right) = 0 \ .$$

No que toca aos extremos, derivando o polinómio $T_n(x)$ obtém-se

$$T_n^{'}(x) = \frac{n}{\sqrt{1-x^2}}\sin(n\arccos(x)) \ .$$

Para os valores de $x_k^{'}$, pertencentes ao intervalo $[-1,1]$, dados pela expressão (2.24), exceptuando os correspondentes a $k=0$ e $k=n$, verifica-se que

$$T_n^{'}\left(x_k^{'}\right)=\frac{n}{\sqrt{1-x^2}}\,\mathbf{sin}\left(n\,\mathbf{arccos}\left(\mathbf{cos}\left(\frac{k\pi}{n}\right)\right)\right)=\frac{n}{\sqrt{1-x^2}}\,\mathbf{sin}\left(k\pi\right)=0.$$

Quanto ao valor que o polinómio T_n toma nos extremos $x_k^{'}$, verifica-se que

$$T_n\left(x_k^{'}\right)=\mathbf{cos}\left(n\,\mathbf{arccos}\left(x_k^{'}\right)\right)=\mathbf{cos}\left(n\,\mathbf{arccos}\left(\mathbf{cos}\left(\frac{k\pi}{n}\right)\right)\right)=\mathbf{cos}\left(k\pi\right)=\left(-1\right)^k.$$

Verifica-se ainda que nos extremos do intervalo $[-1,1]$, $\left|T_n\left(\pm 1\right)\right|=1$, pelo que, estes pontos também são extremos de T_n, todos com o mesmo valor absoluto. ■

Tendo em conta a relação de recorrência $T_{n+1}\left(x\right)=2xT_n\left(x\right)-T_{n-1}\left(x\right)$, é possivel provar, por indução matemática, que $T_n\left(x\right)=2^{n-1}x^n+termos\ \ de\ \ menor\ \ grau$.

Pelo que, definindo

$$\overline{T}_n\left(x\right)=2^{1-n}T_n\left(x\right)$$

este polinómio é mónico, ou seja,

$$\overline{T}_n\left(x\right)=x^n+termos\ \ de\ \ menor\ \ grau.$$

Designemos por $\overline{P}_n$[-1,1] a classe de polinómios mónicos de grau $\leq n$ no intervalo [-1,1].

Teorema 6

O polinómio $\overline{T}_n$ é, de todos os polinómios em $\overline{P}_n$[-1,1] , o de menor norma, ou seja,

$$\left\|\overline{T}_n\right\|_\infty \leq \left\|p\right\|_\infty,\ \forall p\in\overline{P}_n\text{[-1,1]}\ .$$

A demonstração deste resultado pode ser vista em Pina (1995).

Escolhendo os zeros do polinómio de Chebyshev T_{n+1} como nós de interpolação, pode constatar-se que

$$W_n\left(x_k^{'}\right)=\overline{T}_{n+1}\left(x_k^{'}\right)=2^{-n},$$

pelo que,

$\|W_n\|_\infty = 2^{-n}$.

Donde, resulta que

$$e_n(x) = \frac{1}{2^n(n+1)!} f^{(n+1)}(\xi) T_{n+1}(x), \qquad \xi \in [-1,1]. \tag{2.26}$$

Tendo em conta que o valor absoluto máximo dos polinómios de Chebyshev é igual a 1, quando se escolhe os zeros do polinómio de Chebyshev T_{n+1} como nós de interpolação, o majorante do erro de interpolação é dado por

$$\|e_n\|_\infty \le \frac{1}{2^n(n+1)!} \|f^{(n+1)}\|_\infty \quad em \quad [-1,1] \tag{2.27}$$

ou

$$\|e_n\|_\infty \le \frac{(b-a)^{n+1}}{2^{2n+1}(n+1)!} \|f^{(n+1)}\|_\infty \quad em \quad [a,b] . \tag{2.28}$$

A partir da expressão anterior pode mostrar-se que, se as derivadas da função f forem limitadas para qualquer ordem n, a interpolação com nós de Chebyshev é convergente. No entanto, esta proposição não é geralmente verdadeira, como já tínhamos visto anteriormente através da interpolação com nós equidistantes no exemplo do fenómeno de Runge.

Exemplo 6

Considere-se ainda o caso da aproximação da função de Runge por interpolação polinomial usando nós de Chebyshev.

Para o efeito, sejam os seguintes 5 nós de Chebyshev, dados por

$$x_k = \cos\left(\frac{2k-1}{10}\pi\right) \qquad k = 1,...,5 ,$$

cujos valores são

$x_1 = 0.951565163$

$x_2 = 0.5877852522$

$x_3 = 0$

$x_4 = -0.5877852522$

$x_5 = -0.951565163$

Na figura 4, estão representados, a função de Runge, $f(x) = \frac{1}{1+25x^2}$, $x \in [-1,1]$ e os polinómios interpoladores $P_4(x)$ e $Q_4(x)$ obtidos a partir de 5 nós, o primeiro, com nós equidistantes, enquanto o segundo com os nós de Chebyshev considerados.

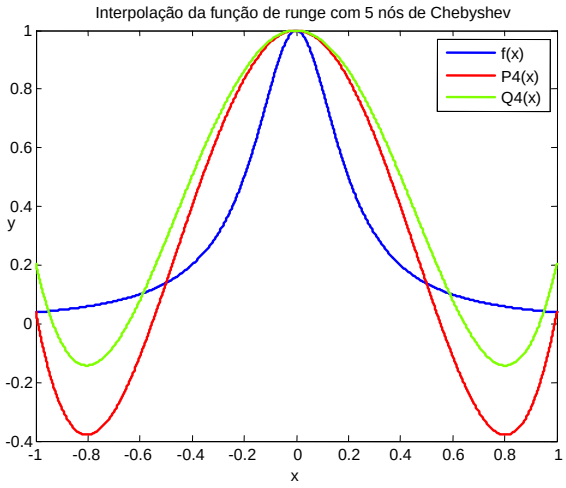


Figura 4 - Interpolação da função de Runge usando 5 nós de Chebyshev.

Analogamente, tomando 10 nós de Chebyshev, ou seja, com

$$x_k = \cos\left(\frac{2k-1}{20}\pi\right) \qquad k = 1,\ldots,10,$$

obtêm-se os gráficos da figura 5, onde se faz a comparação entre a função de Runge, $f(x)$, o polinómio interpolador em 10 nós equidistantes no intervalo $[-1,1]$, P_9, e o polinómio interpolador nos 10 nós de Chebyshev, Q_9.

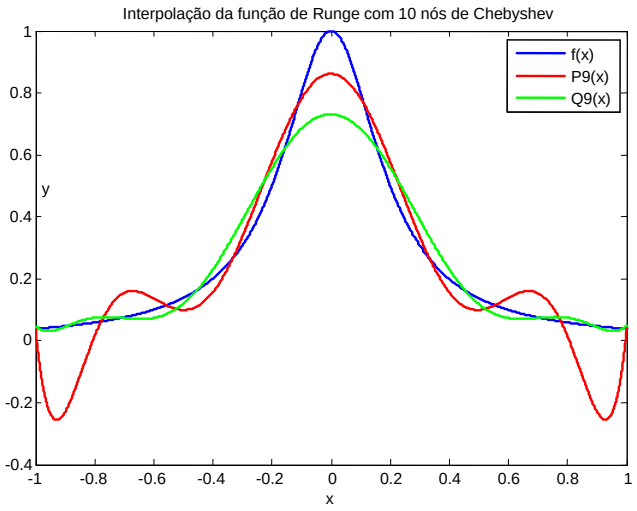


Figura 5 - Interpolação da função de Runge usando 10 nós de Chebyshev.

Como se pode verificar, a utilização dos nós de Chebyshev como nós de interpolação atenua as oscilações, contribuindo para uma diminuição substancial do erro cometido, nomeadamente, nos extremos do intervalo de integração.

Comparando os polinómios $Q_4(x)$ e $Q_9(x)$ constata-se que, o aumento do grau do polinómio interpolador usando nós de Chebyshev, contribui para uma atenuação das oscilações e consequente diminuição do erro de interpolação. Esta constatação prática está de acordo com o que foi afirmado anteriormente quanto à convergência do polinómio interpolador no caso da função interpolada ter derivadas de qualquer ordem limitadas, ou seja, que o erro de interpolação com nós de Chebyshev tende para zero quando o grau n do polinómio interpolador tende para infinito.

2.5. Polinómio interpolador de Hermite

Quando, além dos valores da função, também se tem acesso aos valores das derivadas da função nos nós de interpolação, pode construir-se um polinómio aproximante mais rico em informação, conhecido como polinómio interpolador de Hermite. A partir deste polinómio é possível estimarem-se, não só valores para a função, como também para a sua derivada.

Suponhamos que se pretende interpolar os valores da função e da sua primeira derivada. Ou seja, pretende-se construir um polinómio que interpole os valores da função f e da sua derivada f' nos nós distintos, $x_0, x_1, ..., x_n$.

O polinómio interpolador de Hermite, de grau $2n+1$, H_{2n+1}, terá então de satisfazer as seguintes $2(n+1)$ condições

$$H_{2n+1}(x_i) = f(x_i) \quad e \quad H'_{2n+1}(x_i) = f'(x_i), \qquad i = 0,1,...,n \ . \tag{2.29}$$

Teorema 6 (Polinómio de Hermite)

Consideremos $f(x) \in C^1[a,b]$ e, $x_0, x_1, ..., x_n \in [a,b]$, $n+1$ nós distintos. Então, o único polinómio de grau menor ou igual a $2n+1$, detonado por H_{2n+1}, satisfazendo as condições $H_{2n+1}(x_i) = f(x_i) \quad e \quad H'_{2n+1}(x_i) = f'(x_i), \qquad i = 0,1,...,n$ é dado por

$$H_{2n+1}(x) = \sum_{i=0}^{n} U_i(x) f(x_i) + \sum_{i=0}^{n} V_i(x) f'(x_i) \tag{2.30}$$

onde

$$U_i(x) = \left[1 - 2l'_i(x_i)(x - x_i)\right][l_i(x)]^2 \tag{2.31}$$

e

$$V_i(x) = (x - x_i)[l_i(x)]^2 \quad . \tag{2.32}$$

Recordamos que os polinómios de Lagrange, $l_i(x)$ são obtidos pela expressão (2.6).

Para a construção do polinómio de Hermite, H_{2n+1}, na forma de Newton, consideramos que os nós $x_0, x_1, ..., x_n$ são substituídos pelos nós $x_0, x'_0, x_1, x'_1, ..., x_n, x'_n$ e fazemos x'_k tender para x_k, $k = 0,1,...,n$.

Deste modo,

$$f[x_k, x_k] = \lim_{x'_k \to x_k} f[x'_k, x_k] = \lim_{x'_k \to x_k} \frac{f(x_k) - f(x'_k)}{x_k - x'_k} = f'(x_k) \tag{2.33}$$

O polinómio de Hermite na forma de Newton, será dado por

$$\begin{aligned} H_{2n+1}(x) = {} & f(x_0) + f[x_0, x_0](x - x_0) + f[x_0, x_0, x_1](x - x_0)^2 + \\ & + f[x_0, x_0, x_1, x_1](x - x_0)^2 (x - x_1) + \ldots + \\ & + f[x_0, x_0, x_1, x_1, \ldots, x_n, x_n](x - x_0)^2 (x - x_1)^2 \ldots (x - x_{n-1})^2 (x - x_n) \end{aligned} \tag{2.34}$$

Teorema 7 (Erro de interpolação)

Seja $f \in C^{2n+2}[a,b]$ e, $x_0, x_1, \ldots, x_n$ nós distintos pertencentes ao intervalo $[a,b]$. Então, para qualquer $x^* \in [a,b]$, existe $\xi \in (a,b)$, tal que

$$\begin{aligned} e_{2n+1}(x^*) &= f(x^*) - H_{2n+1}(x^*) \\ &= \frac{f^{(2n+2)}(\xi)}{(2n+2)!} W_n^2(x^*) \end{aligned} \tag{2.35}$$

Exemplo 7

Dada a função, $f(x) = \ln(x) + 1$, calcular, usando a interpolação cúbica, o valor aproximado de $f(1.5)$, bem como o respectivo erro cometido, sabendo que:

$$\begin{array}{ll} f(1) = 1 & f'(1) = 1 \\ f(2) = 1.69314 \quad \text{e} & f'(2) = 0.5 \end{array}$$

Pela expressão (2.34), o polinómio de Hermite que interpola a função e a sua primeira derivada é dado por

$$H_3(x) = f(x_0) + f[x_0, x_0](x - x_0) + f[x_0, x_0, x_1](x - x_0)^2 + f[x_0, x_0, x_1, x_1](x - x_0)^2 (x - x_1).$$

Os coeficientes $f[x_0, x_0]$, $f[x_0, x_0, x_1]$ e $f[x_0, x_0, x_1, x_1]$ serão determinados com recurso a uma tabela de diferenças divididas com repetição dos nós de interpolação.

x	$f(x)$	1ª ordem	2ª ordem	3ª ordem
1	1			
		$\lim_{x'_0 \to x_0} \frac{f(x_0) - f(x'_0)}{x_0 - x'_0} = 1$		
1	1		-0.30686	
		$\frac{1.69314 - 1}{2 - 1} = 0.69314$		0.12372
2	1.69314		-0.18314	
		$\lim_{x'_1 \to x_1} \frac{f(x_1) - f(x'_1)}{x_1 - x'_1} = 0.5$		
2	1.69314			

Tabela 3 - Tabela de diferença divididas para a interpolação de Hermite.

Donde

$$H_3(x) = 1 + (x-1) - 0.30686(x-1)^2 + 0.12372(x-1)^2(x-2),$$

e, portanto, obtém-se $f(1.5) \approx H_3(1.5) = 1.40782$.

Para simulação e experimentação numérica deste tipo de interpolação polinomial em MatLab é utilizada a rotina **hermite** apresentada a seguir, a qual faz também uso da rotina **hermitecoef**.

```
function p = hermite(t,x,y,dy)

%Entrada - t é um vector que indica a(s) abcissa(s) onde
%         queremos calcular o(s) valor(es) de H2n+1(x)
%        - x é um vector que contém os nós
%        - y é um vector que contém os valores nodais
%        - dy é um vector que contém o valor das derivadas
%         nos nós
%Saída   - p é um vector que contém o(s) valor(es) de H2n+1(x)
%         calculado(s) na(s) abcissa(s) de t

a = hermitecoef(x,y,dy);

n = length(x);
for i = 1 : n
  xx(2*i-1) = x(i);
```

```
  xx(2*i) = x(i);
end
for i = 1 : length(t)
  ddd(1) = 1;
  c(1) = a(1);
  for j = 2 : 2*n
    ddd(j) = (t(i) - xx(j-1)).*ddd(j-1);
    c(j) = a(j).*ddd(j);
  end
  p(i) = sum(c);
end


function a = hermitecoef(x,y,dy)

%Entrada - x é um vector que contém os nós
%       - y é um vector que contém os valores nodais
%       - dy é um vector que contém o valor das derivadas
%         nos nós
%Saída   - a é um vector que contém as diferenças divididas
%         que figuram no polinómio interpolador de Hermite

n = length(x);
a(1) = y(1);

% duplicação dos nós e dos valores nodais

for i = 1 : n
  xx(2*i-1) = x(i);
  yy(2*i-1) = y(i);
  xx(2*i) = x(i);
  yy(2*i) = y(i);
end

% cálculo das diferenças divididas de primeira ordem

for k = 1 : n-1
  d(2*k-1,1) = dy(k);
  d(2*k,1) = (yy(2*k+1) - yy(2*k))/(xx(2*k+1) - xx(2*k));
end
d(2*n-1,1) = dy(n);

% cálculo das diferenças divididas de ordem j>1

for j = 2 : 2*(n-1)
  for k = 1 : 2*n-j
    d(k,j) = (d(k+1,j-1) - d(k,j-1))/(xx(k+j) - xx(k));
  end
end
```

```
d(1,2*n-1) = (d(2,2*(n-1)) - d(1,2*(n-1)))/(xx(2*n) - xx(1));

disp('Tabela das diferenças divididas')
disp(d)


for j = 2 : 2*n
  a(j) = d(1,j-1);
end
```

Usando a rotina **hermite**, obtém-se o seguinte:

```
>> hermite(1.5,[1,2],[1,1.69314],[1,0.5])
Tabela das diferenças divididas
   1.0000   -0.3069    0.1137
   0.6931   -0.1931         0
   0.5000         0         0
ans =
   1.4091
```

Pela expressão (2.35), tem-se

$$e_3(1.5) = f(1.5) - H_3(1.5)$$
$$= \frac{f^{(4)}(\xi)}{4!}(1.5-1)^2(1.5-2)^2 \qquad \xi \in (1,2)$$

Tendo em conta o valor máximo de $|f^{(4)}(\xi)|$, $\xi \in (1,2)$, é possível obter o majorante do erro de interpolação cometido, dado por

$$|e_3(1.5)| \le \max_{\xi\in(1,2)} |f^{(4)}(\xi)| \frac{1}{4!} 0.5^4 \quad .$$

Como $\max_{\xi\in(1,2)} |f^{(4)}(\xi)| = \max_{\xi\in(1,2)} \left| -\frac{6}{\xi^4} \right| = 6$,

tem-se

$$|e_3(1.5)| \le 0.15625 \times 10^{-1} \ .$$

De seguida, vamos aproximar a função de Runge no intervalo [-1,1] que temos vindo a considerar, usando interpolação de Hermite com os 5 nós de Chebyshev já utilizados no exemplo 6. Deste modo, pretende-se construir um polinómio de Hermite a partir do seguinte suporte de interpolação:

$x_1 = 0.9515651630$	$f(x_1) = 0.0423067206$	$f'(x_1) = -0.0851583552$
$x_2 = 0.5877852522$	$f(x_2) = 0.1037636361$	$f'(x_2) = -0.3164310121$
$x_3 = 0$	$f(x_3) = 1$	$f'(x_3) = 0$
$x_4 = -0.5877852522$	$f(x_4) = 0.1037636361$	$f'(x_4) = 0.3164310121$
$x_5 = -0.9515651630$	$f(x_5) = 0.0423067206$	$f'(x_5) = 0.0851583552$

Recorrendo ao MatLab com a rotina **hermite**, poder-se-á construir o polinómio interpolador de Hermite, $H(x)$, neste caso, de grau nove, podendo, depois, obter-se o gráfico representado na figura 6, onde se mostra a função de Runge, $f(x)$, e o referido polinómio, $H(x)$.

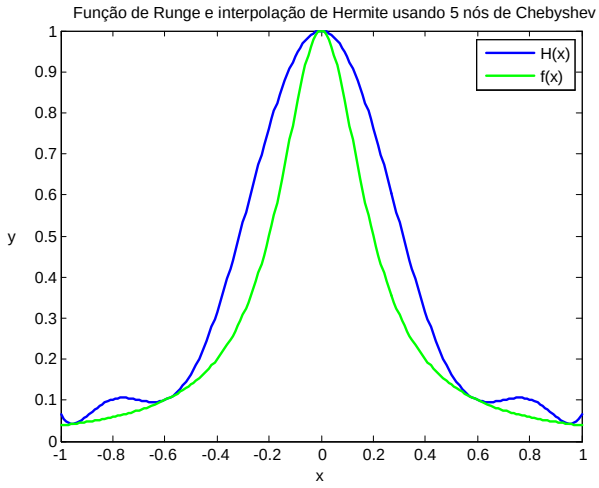


Figura 6 - Interpolação da função de Runge por um polinómio de Hermite.

Comparando este polinómio de Hermite, $H(x)$, com aquele que foi obtido na figura 4 com o mesmo número de nós de Chebyshev, $Q_4(x)$, pode observar-se que $H(x)$ desenvolve menos oscilações, aproximando melhor f do que $Q_4(x)$, conforme seria de esperar, pois, a construção de $H(x)$ inclui a informação adicional das derivadas da função de Runge, f.

3. Aproximação numérica usando Splines

3.1. Introdução

O problema da construção de curvas de formas livres é um problema antigo que surgiu principalmente nas áreas da construção naval e arquitectura, onde o objectivo principal é construir curvas suaves mais conhecidas por Splines, ou seja, curvas compostas por diferentes polinómios interpoladores do mesmo grau (habitualmente, cúbicos) unidos por pontos de controlo (nós de interpolação) nos quais se devem verificar condições de regularidade (continuidade, habitualmente, até à derivada de ordem 3), por forma a obter-se uma aproximação numérica suave (sem variações acentuadas) num dado intervalo de interpolação. Esta técnica também é conhecida como interpolação segmentada, uma vez que cada um dos polinómios que compoêm a curva do Spline é interpolante num subintervalo (segmento) do intervalo de interpolação.

Como já vimos, um aumento do número de nós de interpolação, ou seja, um aumento do grau do polinómio interpolador, nem sempre melhora a aproximação desejada. Recorrendo à interpolação por Splines pode conseguir-se um melhoramento significativo da qualidade dessa aproximação.

De uma forma prática e, atendendo também à origem do termo em inglês, Spline, este pode ser considerado como uma régua de madeira que é utilizada para construir uma curva suave que acompanha a variabilidade dos nós de interpolação dados. Constrói-se, assim, uma função Spline interpoladora com uma certa regularidade, que está intimamente ligada à flexibilidade daquela régua.

3.2. Splines: definição e construção

Em 1946 o matemático Schoenberg apresentou a primeira definição rigososa deste tipo de funções, a partir da qual se chegou à definição actual de Splines.

No desenvolvimento teórico que a seguir se apresenta, divide-se o intervalo $I=[a,b]$ em n subintervalos $[x_0,x_1]$, $[x_1,x_2],\ldots,[x_{n-1},x_n]$, onde $a=x_0<x_1<x_2<\ldots<x_n=b$. Além disso, consideram-se ainda as seguintes notações

$$h=\max_{1\le i\le n} h_i \qquad \text{com} \qquad h_i=x_i-x_{i-1}\ , \tag{3.1}$$

onde h é designado por parâmetro da malha.

Definição 5 (Spline). Uma função Spline interpoladora, $S(x)$, de grau k, no intervalo $[a,b]$, contendo os nós $x_0,x_1,\ldots,x_n$, tais que, $a=x_0<x_1<\ldots<x_n=b$, satisfaz as seguintes condições:

1. $S(x_i)=f(x_i)=yi, \quad i=0,1,\ldots,n;$ (3.2)
2. Em cada subintervalo $[x_{i-1},x_i]$, $\quad i=1,\ldots,n$, $S(x)$ coincide com um polinómio $S_i(x)$ de grau k;
3. $S(x)$ possui derivadas contínuas até a ordem $k-1$ no intervalo $]a,b[$, ou seja, $S(x)\in C^{k-1}(a,b)$.

Relativamente às condições da definição anterior, note-se que a primeira (1) garante que a função $S(x)$ é interpoladora no suporte de pontos considerado, enquanto a segunda (2) implica que os polinómios $S_i(x)$, todos do mesmo grau, têm de satisfazer as condições de interpolação resultantes de (1) e, além disso, por (3) também têm de garantir condições de regularidade (continuidade) até à ordem $k-1$ nos nós intermédios de interpolação (ou seja, em todos os nós excepto nos nós extremos do intervalo de interpolação).

Daremos maior ênfase aos Splines cúbicos (de grau $k=3$) por serem os que, na prática, são de maior utilidade, pois, com eles conseguem-se construir curvas com razoável regularidade (de ordem dois) que fornecem boas aproximações numéricas. No entanto, para se entender melhor a sua construção começaremos por fazer uma breve abordagem aos Splines de grau mais baixo (zero, um e dois).

3.2.1. Splines de grau zero, um e dois

No caso de um Spline de grau zero (figura 7), $k = 0$, $S(x)$ coincide em cada subintervalo do intervalo $[a,b]$ com um polinómio, $S_i(x)$, de grau zero. Assim, em cada subintervalo $[x_{i-1}, x_i[$, $i = 1,...,n$, $S_i(x)$ é tal que

$$S_i(x) = y_i, \quad x_{i-1} \le x < x_i, \quad i = 1,2,...,n \tag{3.3}$$

Teorema 8

Seja $f \in C^1[a,b]$ e S um spline de grau zero. Então, o erro cometido quando se interpola a função f pelo spline S é tal que

$$|e(x)| = |f(x) - S(x)| \le \max_{a \le x \le b} |f'(x)| h \ .$$

Aplicando o teorema 4 do erro de interpolação a um subintervalo genérico $[x_{i-1}, x_i]$ a demonstração deste resultado é imediata.

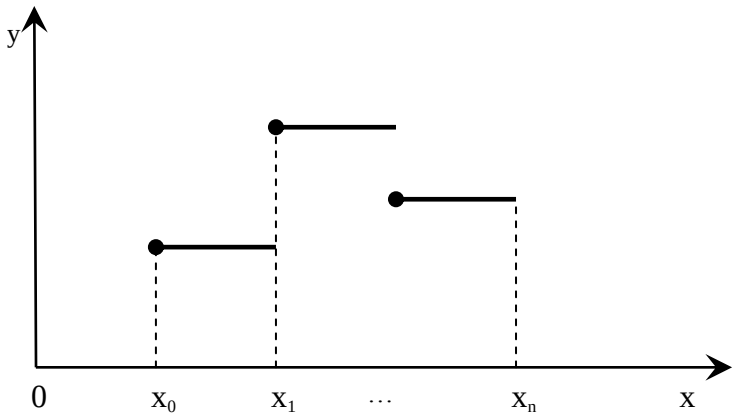


Figura 7 – Função Spline de grau zero.

Tendo em conta a definição 5, uma função Spline de grau um ($k = 1$), $S(x)$, é contínua em todo o intervalo $]a,b[$ e em cada subintervalo coincide com um polinómio $S_i(x)$ de grau um (figura 8). Para garantir a continuidade da função S, cada um dos polinómios S_i pode ser definido da seguinte forma

$$S_i(x) = \frac{x - x_i}{x_{i-1} - x_i} y_{i-1} - \frac{x - x_{i-1}}{x_{i-1} - x_i} y_i, \qquad x_{i-1} \le x \le x_i, \qquad i = 1,2,...,n \tag{3.4}$$

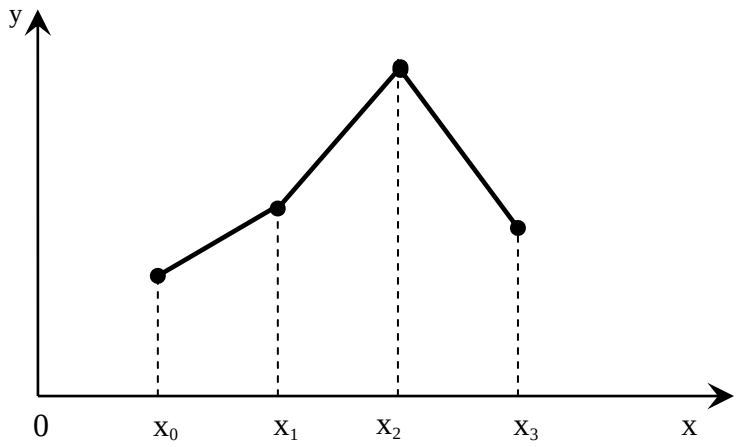


Figura 8 – Spline de grau 1.

Teorema 9

Seja $f \in C^2[a,b]$ e S um spline de grau um. Então, o erro cometido quando se interpola a função f pelo spline S é majorado por

$$|e(x)| = |f(x) - S(x)| \le \frac{1}{8} \max_{a \le x \le b} |f''(x)| h^2 .$$

A demonstração deste resultado faz-se por um racíocinio análogo à do teorema anterior.

Um Spline de grau dois ou quadrático, S, coincide em cada subintervalo $[x_{i-1}, x_i]$ com um polinómio, $S_i(x)$, de grau 2. Além disso, pelas condições de regularidade (3), $S(x)$ tem de ser uma função continuamente diferenciável em todo o intervalo $]a,b[$. Assim, um Spline quadrático é uma curva suave composta por parábolas, $S_i(x)$, que, nos nós de interpolação intermédios, se unem de modo contínuo e com tangentes também contínuas (regularidade até às derivadas de ordem 1) de um subintervalo para o outro.

Como em cada um dos n subintervalos, o Spline é representado por um polinómio interpolador de grau dois, torna-se necessário calcular três coeficientes para definir cada um desses n polinómios (parábolas). Ou seja, a construção do Spline quadrático requer o cálculo de $3n$ coeficientes.

Ora, aplicando as condições de interpolação (1) nos nós x_{i-1} e x_i, resultam duas equações por cada um dos n subintervalos, perfazendo $2n$ equações no intervalo $[a,b]$.
Além disso, pelas condições (3), a continuidade da primeira derivada nos nós intermédios $x_1, x_2, ..., x_{n-1}$, fornece mais $n-1$ equações. Assim, obtêm-.se no total $2n+n-1=3n-1$ equações.

Donde, o número de equações é inferior ao número de coeficientes a serem determinados, pelo que, é necessário impor uma condição suplementar para que se possam determinar os n polinómios de grau dois que definem o Spline quadrático. Uma hipótese é, por exemplo, considerar uma condição na primeira derivada num dos nós extremos, $S'(x_0)$ ou $S'(x_n)$, impondo um determinado valor para essa derivada, no caso desse valor não ser conhecido.

Na construção do Spline quadrático iremos considerar que a derivada no nó x_0 assume um determinado valor m_0. Como em cada subintervalo $[x_{i-1}, x_i]$, S_i é um polinómio de grau dois, pode assumir-se que a sua primeira derivada varia de uma forma linear, ou seja,

$$S_i'(x) = \frac{x_i - x}{h_i} m_{i-1} + \frac{x - x_{i-1}}{h_i} m_i \qquad i = 1,2,...,n \ , \tag{3.5}$$

onde h_i é dado na expressão (3.1) e os parâmetros m_i representam as primeiras derivadas dos polinómios S_i nos nós x_i, tais que

$$m_i = S_i'(x_i) = S'(x_i), \qquad i = 1,...,n \quad \text{e} \quad S'_1(x_0) = S'(x_0) = m_0 \ .$$

A expressão (3.5) representa um polinómio de grau um cuja forma garante a continuidade de $S'(x)$ em $]x_0, x_n[$. Deste modo, integrando (3.5), obtém-se

$$S_i(x) = -\frac{(x_i - x)^2}{2h_i} m_{i-1} + \frac{(x - x_{i-1})^2}{2h_i} m_i + c_i \ .$$

Em cada subintervalo $[x_{i-1}, x_i]$, a constante real c_i, será calculada tendo em conta a condição de interpolação

$$S_i(x_{i-1}) = S(x_{i-1}) = y_{i-1},$$

a qual permite deduzir

$$S_i(x) = y_{i-1} + (x - x_{i-1})\left(1 - \frac{x - x_{i-1}}{2h_i}\right)m_{i-1} + \frac{(x - x_{i-1})^2}{2h_i}m_i, \quad i = 1,2,...,n. \tag{3.6}$$

Atendendo às condições (1) e (2) da definição 5 de Spline interpolador, ainda é necessário garantir que $S_i(x_i) = S(x_i) = y_i$, vindo

$$S_i(x_i) = y_{i-1} + \frac{h_i}{2}(m_{i-1} + m_i) = y_i,$$

donde, resulta que

$$m_i = 2\frac{(y_i - y_{i-1})}{h_i} - m_{i-1}, \qquad i = 1,2,..n. \tag{3.7}$$

Assim, se o valor da derivada de S_1 no nó x_0, m_0, for dado, obtêm-se pela expressão (3.7) os valores de $m_1, m_2, ..., m_n$, pelo que, pela expressão (3.6), o Spline quadrático fica completamente determinado.

Exemplo 8

Determinar a função Spline S_q de grau dois, que interpola os seguintes valores

$y_0 = f(x_0) = f(1) = 2$, $y_1 = f(x_1) = f(3) = -2$, $y_2 = f(x_2) = f(4) = 4$,

$y_3 = f(x_3) = f(5) = -3$, sabendo que $m_0 = f'(x_0) = f'(1) = 0$.

Atendendo aos dados fornecidos, subdivide-se o intervalo $[a,b] = [1,5]$ em três subintervalos, $[1,3]$, $[3,4]$ e $[4,5]$. Deste modo, o Spline quadrático S_q será constituído pelos polinómios de grau dois, S_1, S_2 e S_3, sendo definido da seguinte forma

$$S_q(x) = \begin{cases} S_1(x), & x \in [1,3] \\ S_2(x), & x \in [3,4] \\ S_3(x), & x \in [4,5] \end{cases}$$

Pela expressão (3.7) calculamos os valores das derivadas de S_1, S_2 e S_3, respectivamente, nos nós $x_1 = 3$, $x_2 = 4$ e $x_3 = 5$, ou seja,

$$m_1 = 2\frac{y_1 - y_0}{h_1} - m_0 = 2\frac{-2-2}{3-1} - 0 = -4,$$

$$m_2 = 2\frac{y_2 - y_1}{h_2} - m_1 = 2\frac{4-(-2)}{4-3} - (-4) = 16$$

e

$$m_3 = 2\frac{y_3 - y_2}{h_3} - m_2 = 2\frac{-3-4}{5-4} - 16 = -30\,.$$

Com estes valores, recorrendo à expressão (3.6), determinam-se os polinómios quadráticos que definem $S_q(x)$ cujo gráfico é dado na figura 9, obtendo-se

$$\begin{aligned} S_1(x) &= y_0 + (x - x_0)\left(1 - \frac{x - x_0}{2h_1}\right)m_0 + \frac{(x - x_0)^2}{2h_1} m_1 \\ &= 2 - (x-1)^2 \end{aligned}$$

$$\begin{aligned} S_2(x) &= y_1 + (x - x_1)\left(1 - \frac{x - x_1}{2h_2}\right)m_1 + \frac{(x - x_1)^2}{2h_2} m_2 \\ &= -2 - 4(x-3) + 10(x-3)^2 \end{aligned}$$

$$\begin{aligned} S_3(x) &= y_2 + (x - x_2)\left(1 - \frac{x - x_2}{2h_3}\right)m_2 + \frac{(x - x_2)^2}{2h_3} m_3 \\ &= 4 + 16(x-4) - 23(x-4)^2 \end{aligned}$$

Donde, tem-se

$$S_q(x) = \begin{cases} 2 - (x-1)^2, & , x \in [1,3] \\ -2 - 4(x-3) + 10(x-3)^2, & x \in [3,4] \\ 4 + 16(x-4) - 23(x-4)^2, & x \in [4,5] \end{cases}$$

Como seria de esperar, pode constatar-se que as condições de continuidade até às derivadas de ordem um são satisfeitas, isto é,

$S_1(3) = S_2(3),\ S_2(4) = S_3(4)$ e $S'_1(3) = S'_2(3)$ e $S'_2(4) = S'_3(4)$.

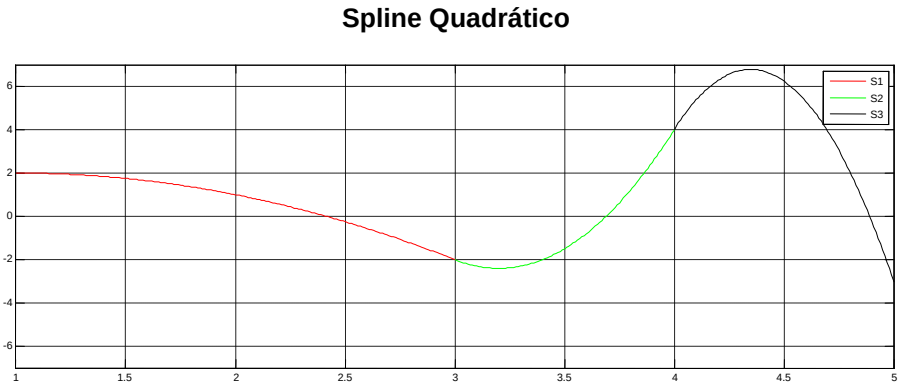


Figura 9 – Gráfico do Spline quadrático $S_q(x)$ do exemplo 8.

3.2.2. Splines cúbicos

Da definição 5, derivam-se as seguintes propriedades do Spline cúbico:

- $S(x_i) = f(x_i) = y_i, \quad i = 0,1,...,n$ (condições de interpolação);
- Em cada subintervalo $[x_{i-1}, x_i]$, $i = 1,2,...,n$, $S(x)$ coincide com um polinómio $S_i(x)$ de grau três;
- $S(x)$, $S'(x)$ e $S''(x)$ são contínuas no intervalo $]a,b[$.

Deste modo, em cada subintervalo $[x_{i-1}, x_i]$, $S_i''(x)$ é um polinómio de grau menor ou igual a um, pelo que, pode ser escrito na forma

$$S_i''(x) = \frac{x_i - x}{h_i} M_{i-1} - \frac{x_{i-1} - x}{h_i} M_i \,, \quad i = 1,2,...,n, \tag{3.8}$$

onde os parâmetros M_i, habitualmente designados por *momentos*, correspondem aos valores das segundas derivadas dos polinómios S_i nos nós x_i, tais que

$$M_i = S_i''(x_i) = S''(x_i), \qquad i = 1,...,n \quad \text{e} \quad S_1''(x_0) = S''(x_0) = M_0 \,.$$

A construção é análoga à do Spline quadrático e, a definição de cada polinómio $S_i''(x)$ através de (3.8), garante a continuidade de $S''(x)$ em $]x_0, x_n[$. Integrando duas vezes (3.8), obtém-se

$$S_i(x) = \frac{(x_i - x)^3}{6h_i} M_{i-1} - \frac{(x_{i-1} - x)^3}{6h_i} M_i + c_i x + d_i \ ,$$

que, depois de alguns ajustes, se pode escrever na forma

$$S_i(x) = \frac{(x_i - x)^3}{6h_i} M_{i-1} - \frac{(x_{i-1} - x)^3}{6h_i} M_i + \frac{x_i - x}{h_i} c_i - \frac{x_{i-1} - x}{h_i} d_i \ . \tag{3.9}$$

As constantes de integração, c_i e d_i, são determinadas, tendo em conta as condições de interpolação

$$S_i(x_{i-1}) = y_{i-1}, \qquad S_i(x_i) = y_i \ , \qquad i = 1,2,...,n \ . \tag{3.10}$$

Usando estas condições na expressão (3.9) e, após algumas simplificações, chega-se às seguintes relações

$$c_i = y_{i-1} - \frac{1}{6} M_{i-1} h_i^2 \qquad e \qquad d_i = y_i - \frac{1}{6} M_i h_i^2 \ ,$$

as quais permitem escrever

$$\begin{aligned} S_i(x) = {} & \frac{(x_i - x)^3}{6h_i} M_{i-1} + \frac{(x - x_{i-1})^3}{6h_i} M_i + \\ & + \left(y_{i-1} - \frac{1}{6} M_{i-1} h_i^2 \right) \frac{x_i - x}{h_i} + \left(y_i - \frac{1}{6} M_i h_i^2 \right) \frac{x - x_{i-1}}{h_i} \end{aligned} \ , \ i = 1,2,...,n \ . \tag{3.11}$$

Para terminar a construção do Spline cúbico falta o cálculo dos parâmetros M_i, o qual se torna mais moroso que no caso do Spline quadrático, conforme veremos a seguir.

Para isso, usam-se as condições de continuidade das primeiras derivadas nos nós interiores, ainda não consideradas, ou seja,

$$S_i'(x_i) = S_{i+1}'(x_i), \qquad i = 1,2,...,n-1. \tag{3.12}$$

Ora, derivando (3.11) tem-se

$$S_i'(x) = -\frac{(x_i - x)^2}{2h_i} M_{i-1} + \frac{(x - x_{i-1})^2}{2h_i} M_i + \frac{y_i - y_{i-1}}{h_i} - (M_i - M_{i-1}) \frac{h_i}{6} \ , \tag{3.13}$$

donde,

$$S_i'(x_i) = \frac{y_i - y_{i-1}}{h_i} + \frac{1}{6}h_i M_{i-1} + \frac{1}{3}h_i M_i$$

$$S_{i+1}'(x_i) = \frac{y_{i+1} - y_i}{h_{i+1}} - \frac{1}{3}h_{i+1} M_i - \frac{1}{6}h_{i+1} M_{i+1} \quad .$$

Substituindo as duas últimas igualdades na expressão (3.12) e, após rearranjo de termos, vem

$$\frac{1}{6}h_i M_{i-1} + \frac{h_i + h_{i+1}}{3} M_i + \frac{1}{6}h_{i+1} M_{i+1} = \frac{y_{i+1} - y_i}{h_{i+1}} + \frac{y_{i-1} - y_i}{h_i}, \qquad i = 1,...,n-1. \tag{3.14}$$

A expressão (3.14) conduz a um sistema de $n-1$ equações, o que é insuficiente para determinar as $n+1$ incógnitas, $M_0, M_1, ..., M_n$. Deste modo, torna-se necessário impor duas condições suplementares, as quais dão origem a Splines cúbicos conhecidos, como por exemplo:

- Spline cúbico completo:

  $$S_1'(x_0) = f'(x_0) = y_0', \qquad S_n'(x_n) = f'(x_n) = y_n' \tag{3.15}$$

  Com as derivadas y_0' e y_n' dadas *a priori* e, tendo em conta a expressão (3.13), após algumas simplificações, deduzem-se as duas equações seguintes

  $$\begin{aligned} &\frac{y_1 - y_0}{h_1} - \frac{h_1 M_1}{6} - \frac{h_1 M_0}{3} = y_0' \\ &\frac{y_n - y_{n-1}}{h_n} + \frac{h_n M_{n-1}}{6} + \frac{h_n M_n}{3} = y_n' \end{aligned} \tag{3.16}$$

- Spline cúbico natural:

  $$S_1''(x_0) = M_0 = 0, \qquad S_n''(x_n) = M_n = 0 \tag{3.17}$$

- Spline cúbico periódico:

  $$\begin{aligned} &y_0 = y_n, \\ &S'(x_0) = S'(x_n). \end{aligned} \tag{3.18}$$

Em todos estes casos, o problema do cálculo das $n+1$ incógnitas, $M_0, M_1, ..., M_n$, a partir de $n+1$ equações lineares, pode ser formulado através de um sistema matricial adequado.

Multiplicando (3.14) pelo factor $\frac{6}{h_i + h_{i+1}}$, obtém-se

$$\mu_i M_{i-1} + 2M_i + \lambda_i M_{i+1} = d_i, \qquad i = 1, 2, ..., n-1 \quad , \tag{3.19}$$

onde

$$\mu_i = \frac{h_i}{h_i + h_{i+1}}, \ \lambda_i = \frac{h_{i+1}}{h_i + h_{i+1}} \quad \text{e} \quad d_i = \frac{6}{h_i + h_{i+1}} \left[\frac{y_{i+1} - y_i}{h_{i+1}} + \frac{y_{i-1} - y_i}{h_i} \right].$$

Para o caso do Spline cúbico natural, ou seja, considerando $S_1''(x_0) = M_0 = 0$ e $S_n''(x_n) = M_n = 0$, a determinação das restantes incógnitas $M_1, M_2, ..., M_{n-1}$, pode ser feita de (3.19) recorrendo a um sistema tridiagonal de $n-1$ equações e $n-1$ incógnitas. Na forma matricial, tal sistema é dado por

$$\begin{bmatrix} 2 & \lambda_1 & 0 & 0 & \dots & 0 \\ \mu_2 & 2 & \lambda_2 & 0 & \dots & 0 \\ 0 & \mu_3 & 2 & \lambda_3 & \dots & 0 \\ 0 & 0 & \mu_4 & 2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \lambda_{n-1} \\ 0 & 0 & 0 & & \mu_n & 2 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ \dots \\ M_{n-2} \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \dots \\ d_{n-2} \\ d_{n-1} \end{bmatrix} \quad . \tag{3.20}$$

Ora, facilmente se pode mostrar que a matriz do sistema (3.20) é de diagonal estritamente dominante por linhas, sendo, por isso, invertível, como se demonstra em (Pina, 1995, Cap. 6), donde, o sistema tem solução única, qualquer que seja o segundo membro. Além disso, como a matriz do sistema é tridiagonal, a obtenção dos momentos torna-se uma tarefa relativamente fácil através da aplicação de um algoritmo adequado em MatLab, como por exemplo, o conhecido algoritmo de Thomas para sistemas tridiagonais (Pina, 1995, Cap. 6).

Exemplo 9

Determinar o Spline cúbico natural, $S_c(x)$, que interpola os seguintes valores

$y_0 = f(x_0) = f(1) = 2$, $y_1 = f(x_1) = f(3) = -2$, $y_2 = f(x_2) = f(4) = 4$, $y_3 = f(x_3) = f(5) = -3$.

Como são dados quatro pontos, para determinar $S_c(x)$ é necessário obter três polinómios de grau três, $S_1(x)$, $S_2(x)$ e $S_3(x)$, interpoladores nos intervalos $[1,3]$, $[3,4]$ e $[4,5]$, respectivamente. Além disso, visto que se trata de um Spline cúbico natural, tem-se, $M_0 = M_3 = 0$.

Neste caso, aplicando (3.14), obtém-se o sistema de equações

$$\begin{cases} \frac{1}{6}h_1 M_0 + \frac{h_1+h_2}{3}M_1 + \frac{1}{6}h_2 M_2 = \frac{y_2-y_1}{h_2} + \frac{y_0-y_1}{h_1} \\ \frac{1}{6}h_2 M_1 + \frac{h_2+h_3}{3}M_2 + \frac{1}{6}h_3 M_3 = \frac{y_3-y_2}{h_3} + \frac{y_1-y_2}{h_2} \end{cases}$$

$$\Leftrightarrow \begin{cases} \frac{2+1}{3}M_1 + \frac{1}{6}M_2 = \frac{4-(-2)}{1} + \frac{2-(-2)}{2} \\ \frac{1}{6}M_1 + \frac{1+1}{3}M_2 = \frac{-3-4}{1} + \frac{-2-4}{1} \end{cases}$$

com solução $M_1 = \frac{270}{23}$ *e* $M_2 = -\frac{516}{23}$.

Assim, usando (3.11), podem escrever-se as equações de $S_1(x)$, $S_2(x)$ e $S_3(x)$

$$S_1 = \frac{(x_1-x)^3}{6h_1}M_0 + \frac{(x-x_0)^3}{6h_1}M_1 + \left(y_0 - \frac{M_0}{6}h_1^2\right)\frac{x_1-x}{h_1} + \left(y_1 - \frac{M_1}{6}h_1^2\right)\frac{x-x_0}{h_1}$$

$$S_2 = \frac{(x_2-x)^3}{6h_2}M_1 + \frac{(x-x_1)^3}{6h_2}M_2 + \left(y_1 - \frac{M_1}{6}h_2^2\right)\frac{x_2-x}{h_2} + \left(y_2 - \frac{M_2}{6}h_2^2\right)\frac{x-x_1}{h_2}$$

$$S_3 = \frac{(x_3-x)^3}{6h_3}M_2 + \frac{(x-x_2)^3}{6h_3}M_3 + \left(y_2 - \frac{M_2}{6}h_3^2\right)\frac{x_3-x}{h_3} + \left(y_3 - \frac{M_3}{6}h_3^2\right)\frac{x-x_2}{h_3}$$

e, portanto, substituindo todos os valores conhecidos, após simplificações, determina-se o Spline cúbico, $S_c(x)$, para $x \in [1,5]$, cujo gráfico é dado na figura 10:

$$S_c(x)=\begin{cases}\frac{45}{46}(x-1)^3-\frac{136}{23}x+\frac{182}{23}, & x\in[1,3]\\ \frac{45}{23}(4-x)^3-\frac{86}{23}(x-3)^3+\frac{269}{23}x-\frac{898}{23}, & x\in[3,4]\\ -\frac{86}{23}(5-x)^3-\frac{247}{23}x+\frac{1166}{23}, & x\in[4,5]\end{cases}$$

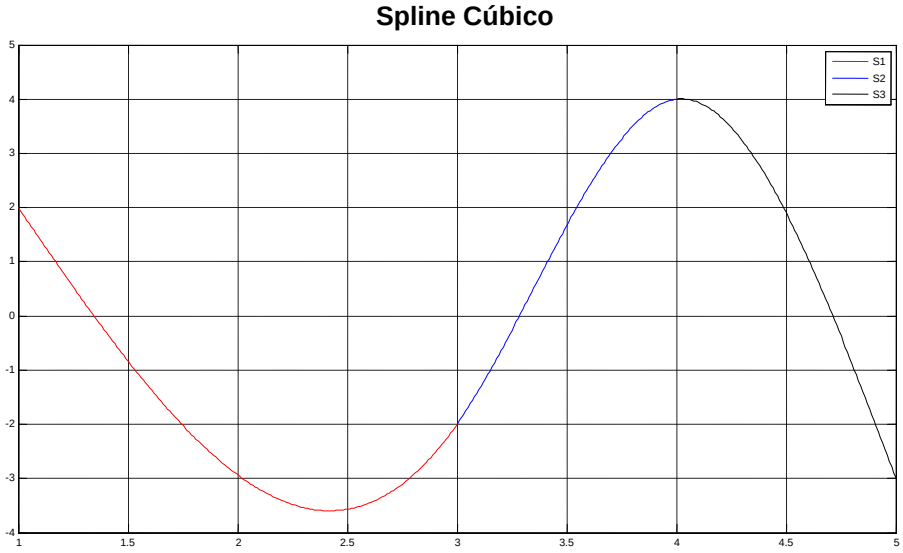


Figura 10 - Spline cúbico natural $S_c(x)$ do exemplo 9.

Teorema 10 - Erro de interpolação de um Spline cúbico (Quarteroni, 2007**)**

Seja $f\in C^4([a,b])$ e o intervalo $[a,b]$ subdividido em intervalos de comprimento h_i, $i=1,2,...,n$, onde $h=\max_i h_i$. Sendo S_c o Spline cúbico interpolante da função f em $[a,b]$, então

$$\|f-S_c\|_\infty \le \frac{5}{384}h^4\|f^{(4)}\|_\infty ,$$

$$\|f'-S_c'\|_\infty \le \frac{1}{24}h^3\|f^{(4)}\|_\infty ,$$

$$\|f''-S_c''\|_\infty \le \frac{3}{8}h^2\|f^{(4)}\|_\infty .$$

Exemplo 10 – Spline cúbico interpolante da função de Runge

Interpola-se a função de Runge, $f(x)=\frac{1}{1+25x^2}$, $x\in[-1,1]$, cuja derivada é $f'(x)=-\frac{50x}{(1+25x^2)^2}$, usando-se um Spline cúbico completo.

Para o efeito, utiliza-se o seguinte algoritmo que inclui a rotina ***spline*** do MatLab, a qual permite obter o Spline cúbico pretendido:

```
x = -1:0.01:1;
y = 1./(1 + 25*x.^2);
dy0=25/338;
dyn=-25/338;
yy=[dy0, y,dyn];
xi = -1:0.01:1;
yi = spline(x, yy, xi);
plot(x, y, 'x', xi, yi), title('Interpolação da função de Runge com spline cúbico')
```

Este algoritmo MatLab permite obter o gráfico da figura 11, o qual contém a função de Runge e o Spline cúbico completo obtido com pontos equidistantes de passo h=0.01.

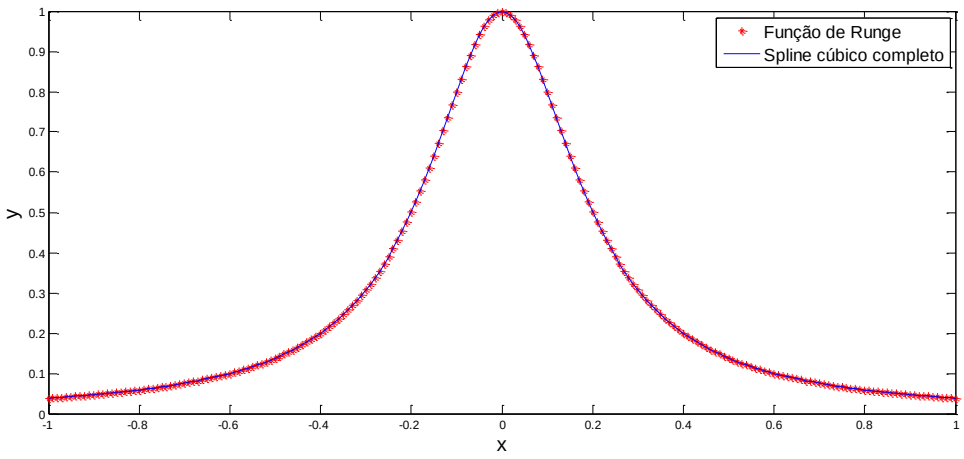


Figura 11 - Spline cúbico completo interpolante da função de Runge, com h=0.01.

Conforme se pode constatar, a aproximação da função de Runge pelo Spline cúbico completo interpolante é bastante razoável. Isso também sucede porque se utilizou um passo h muito pequeno, pois, conforme se pode constatar pelo teorema 10, se a derivada de quarta ordem da função interpolada for limitada, o majorante do erro de interpolação

cometido dependerá apenas do espaçamento máximo h entre os nós considerados. Quanto menor for h, menor será o erro de interpolação cometido (tenderá mesmo para zero quando h também tender para zero).

Na figura 12 pode observar-se o caso da aproximação da função de Runge pelo Spline cúbico completo interpolante com h=0.1.

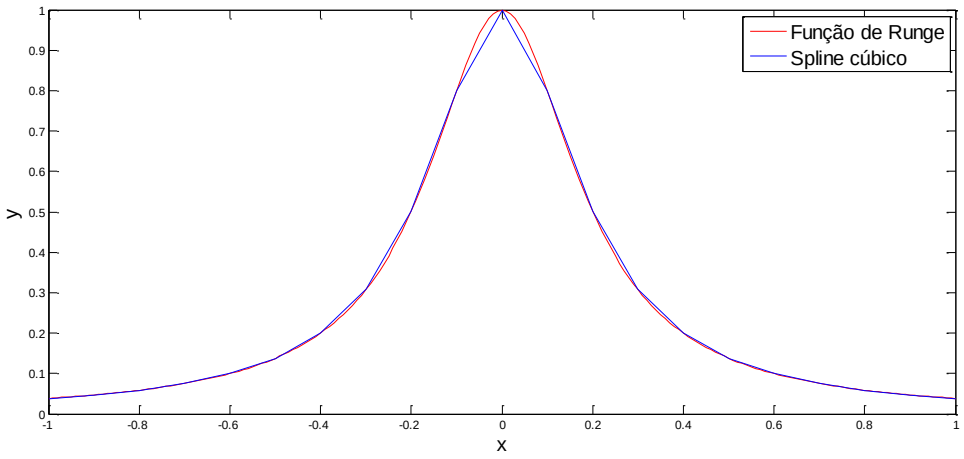


Figura 12 – Spline cúbico completo interpolante da função de Runge, com h=0.1.

Usando 10 nós de interpolação com h=2/9 poder-se-ia verificar, também neste caso, que o Spline cúbico aproxima melhor a função de Runge do que os polinómios interpoladores obtidos nas figuras 5 e 6 do Cap. 2 com 10 nós de Chebyshev.

Para $h = 0.01$ e, tendo em conta que o valor máximo da derivada de quarta ordem da função de Runge, $f(x) = \dfrac{1}{1+25x^2}$, no intervalo $[-1,1]$, é dado por $\max\limits_{-1\le x\le 1} \left| f^{(4)}(x) \right| = 15000$, um majorante do erro de interpolação cometido, em valor absoluto, é, tal que

$$\left\| f - S_c \right\|_\infty \le \frac{5}{384} h^4 \left\| f^{(4)} \right\|_\infty = \frac{5}{384} \times 0.01^4 \times 15000 \approx 0.1953 \times 10^{-5}.$$

Este valor obtido para o majorante do erro de interpolação, confirma a qualidade da aproximação do Spline cúbico interpolante da função de Runge considerada.

3.3. Splines na forma paramétrica

Em certas aplicações, como, por exemplo, no desenvolvimento de protótipos nas indústrias automóvel, de aviação ou naval, surgiu a necessidade de se trabalhar com curvas paramétricas para modelação dos mais variados objectos com formas suaves.

Na representação paramétrica de curvas, o comportamento da curva ao longo do tempo é definido, independentemente, por uma equação para cada um dos eixos.

A forma geral dessas curvas, no caso de grau três (cúbicas), pode ser dada por

$$\begin{cases} x = X(t) = a_x t^3 + b_x t^2 + c_x t + d_x \\ y = Y(t) = a_y t^3 + b_y t^2 + c_y t + d_y \\ z = Z(t) = a_z t^3 + b_z t^2 + c_z t + d_z \end{cases} \tag{3.21}$$

Deste modo, a função Spline cúbica paramétrica dependente das variáveis x, y e z, é da forma

$$S(x,y,z) = \begin{cases} S_1(x,y,z), \\ S_2(x,y,z) \\ \quad \ldots \\ S_n(x,y,z) \end{cases} \tag{3.22}$$

sendo um ponto qualquer da curva, P, definido por

$$P = [X(t), Y(t), Z(t)].$$

Exemplo 11

Construir um Spline cúbico paramétrico que interpola os pontos a, b, c, d, e, f, de $\Re^3$, tal que:

$$a = (0,0,0),\ b = (0,0,2),\ c = (0,2,2),\ d = (2,2,0),\ e = (2,2,2) \quad \text{e} \quad f = (2,0,2).$$

Para o efeito, desenvolveu-se a rotina ***spline_param*** (Spline cúbico tridimensional) a seguir indicada, que, considerando como vectores de entrada, x, y e z, dados por $x = [0,0,0,2,2,2]$, $y = [0,0,2,2,2,0]$ e $z = [0,2,2,0,2,2]$, permite determinar o Spline cúbico paramétrico cujo gráfico é representado na figura 13.

Os vectores de entrada x, y e z representam as coordenadas de cada ponto segundo cada um dos eixos cartesianos.

```
function [xi,yi,zi]=spline_param(x,y,z)
t(1)=0;
for i=1:length(x)-1
   t(i+1)=t(i)+sqrt((x(i+1)-x(i))^2+(y(i+1)-y(i))^2+(z(i+1)-z(i))^2);
end
k=[t(1):(t(length(t))-t(1))/100:t(length(t))];
xi=spline(t,x,k);
yi=spline(t,y,k);
zi=spline(t,z,k);
plot3(xi,yi,zi,'r*')
hold on
plot3(x,y,z)
grid on
```

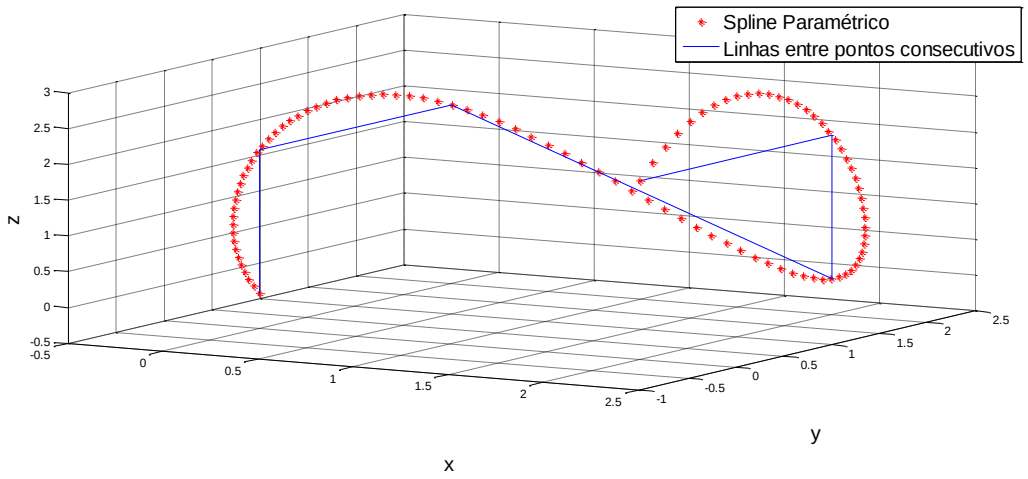


Figura 13 - Spline cúbico paramétrico do exemplo 11.

3.4. Curvas de Bézier

As curvas de Bézier (Su, 1989) foram desenvolvidas na década de 70, na indústria automóvel, pelo engenheiro, Pierre Bézier (1910-1999), da Renault, que as introduziu pela primeira vez no domínio da computação gráfica, para a concepção de carroçarias de automóveis. Estas curvas são construídas com base na aproximação por sucessivos segmentos de recta que unidos formam polígonos. Podem ser lisas, ter curvatura acentuada e formarem laços.

As técnicas utilizadas na construção das curvas de Bézier, constituíram a base da fundação matemática do UNISURF (Bezier, 1986), sistema assistido por computador para design de curvas e superfícies desenvolvido por Bézier em 1971.

Uma curva de Bézier de ordem k é um polinómio de grau $k-1$ definido por k pontos de controlo. O traçado da curva é gerado por equações, onde as variáveis x e y referentes a coordenadas de cada ponto da curva, são dependentes das coordenadas dos vértices do polígono de controlo ou polígono de Bézier, definido pelos pontos de controlo. O polígono de controlo é obtido, unindo os sucessivos pontos de controlo, através de segmentos de recta.

Resumem-se a seguir, as propriedades das curvas de Bézier:

- A curva é determinada, após a definição dos pontos de controlo, e está totalmente contida no polígono convexo definido por esses pontos de controlo;
- O grau do polinómio que define a curva de Bézier, é uma unidade a menos que o número de pontos de controlo;
- A curva interpola o primeiro e o último ponto de controlo;
- No primeiro ponto de controlo, a curva é tangente ao segmento de recta que une o primeiro e o segundo ponto de controlo;
- No último ponto de controlo, a curva é tangente ao segmento de recta que une o penúltimo e o último ponto de controlo;
- A curva tem um comportamento simétrico em relação a t e $(1-t)$, daí que, esta propriedade permite a inversão da ordem dos vértices do polígono de controlo, sem que haja alteração da forma da curva;
- Fazendo coincidir o primeiro com o último ponto de controlo, torna-se possível definir curvas fechadas;
- Se o grau do polinómio da curva de Bézier não for muito elevado, a curva segue de uma forma mais ou menos razoável o polígono de controlo, sem oscilações acentuadas.

Um exemplo de uma curva de Bézier pode ser visualizado na figura 14.

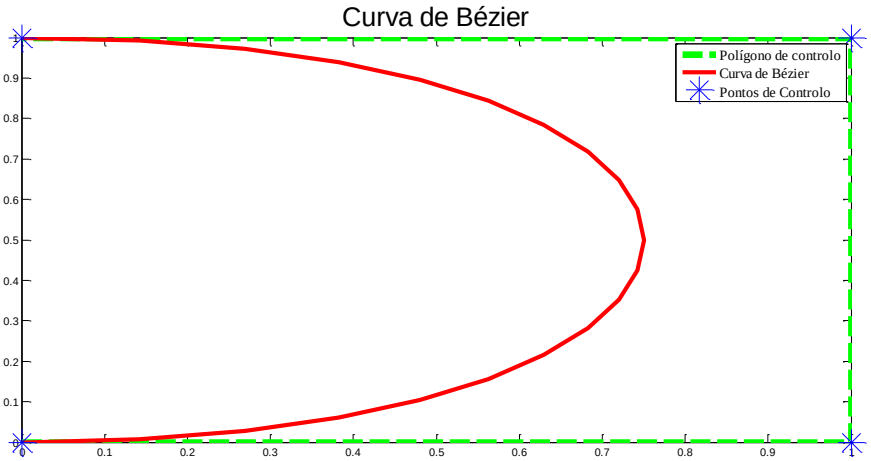


Figura 14 – Exemplo de uma curva de Bézier

Em qualquer instante t, os pontos de controlo influenciam a trajectória da curva, pelo que, não é possível ter um controlo local eficaz, excepto nos extremos.

O grau mais utilizado para as curvas de Bézier é o terceiro, pois, com o grau inferior a 3 temos pouca liberdade de opções e, sendo superior a 3, a curva torna-se muito difícil de modelar.

3.4.1. Definição da curva de Bézier

Dados os vértices do polígono de controlo, $P_i, i = 0,1,...n$, onde $P_i = (x_i, y_i)$, a curva de Bézier pode ser obtida em função dos designados polinómios de Bernstein (Lorentz, 1953).

Polinómios de Bernstein

Tendo em conta o desenvolvimento binomial,

$$1 = [t + (1-t)]^n = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i}, \tag{3.23}$$

os polinómios de Bernstein de grau n, podem escrever-se como,

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0,1,...,n, \tag{3.24}$$

onde $\binom{n}{i} = \frac{n!}{i!(n-i)!}$.

Em geral, existem $n+1$ polinómios de Bernstein de grau n. Por exemplo, os polinómios de Bernstein de grau 1, 2 e 3 são dados, respectivamente, por

$$B_0^1(t) = 1-t, \qquad B_1^1(t) = t$$

$$B_0^2(t) = (1-t)^2, \qquad B_1^2(t) = 2t(1-t), \qquad B_2^2(t) = t^2$$

$$B_0^3(t) = (1-t)^3, \qquad B_1^3(t) = 3t(1-t)^2, \qquad B_2^3(t) = 3t^2(1-t), \qquad B_3^3(t) = t^3$$

Para 3 pontos de controlo (polinómio de grau 2) representam-se no gráfico da figura 15 os polinómios de Bernstein de grau dois correspondentes, para $t \in [0,1]$.

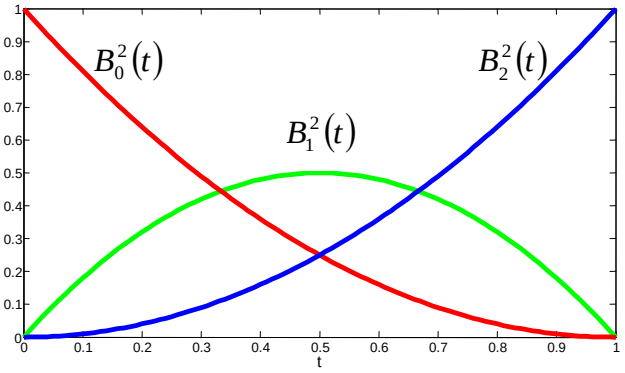


Figura 15 – Polinómios de Bernstein para n=3

Propriedades dos polinómios de Bernstein

- **Relação de recorrência**

Definindo $B_0^0(t) = 1$ e $B_i^n(t) = 0$ para $i < 0$ ou $i > n$, os polinómios de Bernstein podem ser definidos recursivamente da seguinte forma:

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t) \text{ para } i = 1,2,...,n-1 ; \qquad (3.25)$$

- **Não negatividade**

Os polinómios de Bernstein são não negativos no intervalo $[0,1]$, ou seja,

$B_i^n(t) \geq 0, \quad 0 \leq t \leq 1;$

- **Formam uma partição da unidade**

$$\sum_{i=0}^{n} B_i^n(t) = 1, \qquad \forall t \in R;$$

- **Simetria**

$B_i^n(t) = B_{n-i}^n(1-t)$.

Definição 6 (curvas de Bézier)

Dada uma sucessão de pontos de controlo $\{P_i\}_{i=0,\ldots,n}$, com $P_i = (x_i, y_i)$, a curva de Bézier de grau n, é definida por

$$b_{in}(t) = \sum_{i=0}^{n} P_i B_i^n(t) \tag{3.26}$$

onde, $B_i^n(t)$, para $i = 0,1,\ldots,n$, são as funções peso dadas pelos polinómios de Bernstein de grau n, com $t \in [0,1]$.

A expressão (3.26) pode ser parametrizada da seguinte forma

$b_{in}(t) = (x_{in}(t), y_{in}(t), z_{in}(t))$, onde

$$x_{in}(t) = \sum_{i=0}^{n} x_i B_i^n(t) \quad , \quad y_{in}(t) = \sum_{i=0}^{n} y_i B_i^n(t), \quad e \quad z_{in}(t) = \sum_{i=0}^{n} z_i B_i^n(t) \quad , \quad 0 \leq t \leq 1$$

Para se poder ter um maior controlo da curva é conveniente proceder-se a uma subdivisão da mesma. Uma forma de proceder a essa subdivisão é utilizar o algoritmo de Casteljau (Paluszny, 2005), que, genericamente, consiste em interpolação linear de interpolações lineares dos sucessivos pontos de controlo, $P_i, i = 0,1,\ldots n$, onde $P_i = (x_i, y_i)$.

De acordo com o algoritmo de Casteljau[3], a curva linear de Bézier obtém-se pela interpolação linear de dois pontos de controlo, P_0 e P_1, ou seja,

[3] Paul de Casteljau (1930-1999), Físico e Matemático da Citroen

$$b_{01}(t) = (1-t)P_0 + tP_1, \text{ definida para } 0 \le t \le 1,$$

a qual resulta num segmento de recta ligando os vértices P_0 e P_1.

A curva quadrática de Bézier consiste na interpolação linear das interpolações lineares obtidas pelos pontos de controlo P_0 e P_1 e por P_1 e P_2. Assim, tais interpolações lineares conduzem a

$$b_{01}(t) = (1-t)P_0 + tP_1, \quad b_{12}(t) = (1-t)P_1 + tP_2,$$

pelo que, a curva quadrática de Bézier é dada por

$$\begin{aligned} b_{02}(t) &= (1-t)b_{01}(t) + tb_{12}(t) \\ &= (1-t)\left[(1-t)P_0 + tP_1\right] + t\left[(1-t)P_1 + tP_2\right] \\ &= (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2 \end{aligned} \tag{3.27}$$

Na forma matricial, a equação anterior pode ser escrita como

$$b_{02}(t) = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \end{bmatrix}, \qquad 0 \le t \le 1 \tag{3.28}$$

Seguindo um raciocínio análogo, a curva cúbica de Bézier construída a partir de um polígono convexo de controlo definido pelos pontos, P_0, P_1, P_2 e P_3, será obtida depois de sucessivas interpolações lineares dos pontos de controlo.

Começando por definir, as três primeiras interpolações lineares

$$b_{01}(t) = (1-t)P_0 + tP_1, \quad b_{12}(t) = (1-t)P_1 + tP_2, \quad b_{23}(t) = (1-t)P_2 + tP_3,$$

podem obter-se, por interpolação linear das anteriores, as curvas quadráticas

$$b_{02}(t) = (1-t)b_{01}(t) + tb_{12}(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

$$b_{13}(t) = (1-t)b_{12}(t) + tb_{23}(t) = (1-t)^2 P_1 + 2t(1-t)P_2 + t^2 P_3$$

e, por conseguinte, a interpolação linear destas, conduz à curva cúbica de Bézier

$$\begin{aligned} b_{03}(t) &= (1-t)b_{02}(t) + tb_{13}(t) \\ &= (1-t)\left[(1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2\right] + t\left[(1-t)^2 P_1 + 2t(1-t)P_2 + t^2 P_3\right] \\ &= (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \end{aligned} \quad (3.29)$$

Esta curva cúbica também pode ser escrita na forma matricial, vindo

$$b_{03}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}, \qquad 0 \le t \le 1 \qquad (3.30)$$

Tendo em conta que cada ponto de controlo, $P_i \ , i = 0,1,2,3$, é dado pelo par ordenado $P_i = (x_i, y_i)$, a curva cúbica de Bézier pode ser expressa na forma paramétrica por

$$\begin{aligned} x(t) &= (1-t)^3 x_0 + 3t(1-t)^2 x_1 + 3t^2(1-t)x_2 + t^3 x_3 \\ y(t) &= (1-t)^3 y_0 + 3t(1-t)^2 y_1 + 3t^2(1-t)y_2 + t^3 y_3 \end{aligned} \quad (3.31)$$

Na figura 16 pode observar-se a construção de uma curva cúbica de Bézier utilizando o algoritmo de Casteljau.

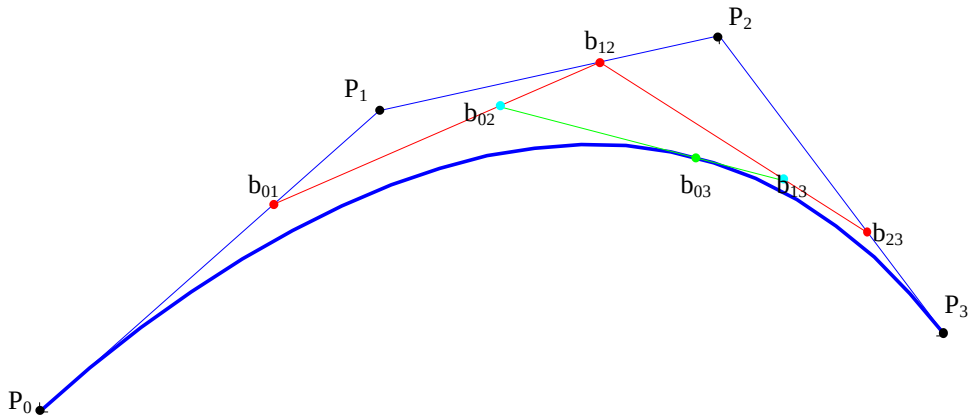


Figura 16 - Curva cúbica de Bézier obtida com o algoritmo de Casteljau com t=1/3.

Exemplo 12

Obter a curva de Bézier definida pelos seguintes pontos de controlo:

$P_0 = (0,2), \qquad P_1 = (1,-2), \qquad P_2 = (3,0), \qquad P_3 = (4,3).$

Tendo em conta que são dados 4 pontos de controlo, a curva será de grau 3, sendo a sua forma paramétrica

$$x_{03}(t) = (1-t)^3 x_0 + 3t(1-t)^2 x_1 + 3t^2(1-t)x_2 + t^3 x_3$$

$$y_{03}(t) = (1-t)^3 y_0 + 3t(1-t)^2 y_1 + 3t^2(1-t)y_2 + t^3 y_3$$

a qual, substituindo as coordenadas dos pontos de controlo, permite obter

$$x_{03}(t) = (1-t)^3 0 + 3t(1-t)^2 1 + 3t^2(1-t)3 + t^3 4 = 3t + 3t^2 - 2t^3$$

$$y_{03}(t) = (1-t)^3 2 + 3t(1-t)^2(-2) + 3t^2(1-t)0 + t^3 3 = 2 - 12t + 18t^2 - 5t^3$$

Na figura 17 faz-se um esboço da respectiva curva de Bezier, bem como do polígono definido pelos pontos de controlo dados, o qual é obtido usando a seguinte sequência algorítmica em MatLab:

```
xi=[0,1,3,4];
yi=[2,-2,0,3];
t=0:0.01:1;
x=3*t.^1+3*t.^2-2*t.^3;
```

```
y=2-12*t.^1+18*t.^2-5*t.^3;
plot(xi,yi)
hold on
plot(x,y)
```

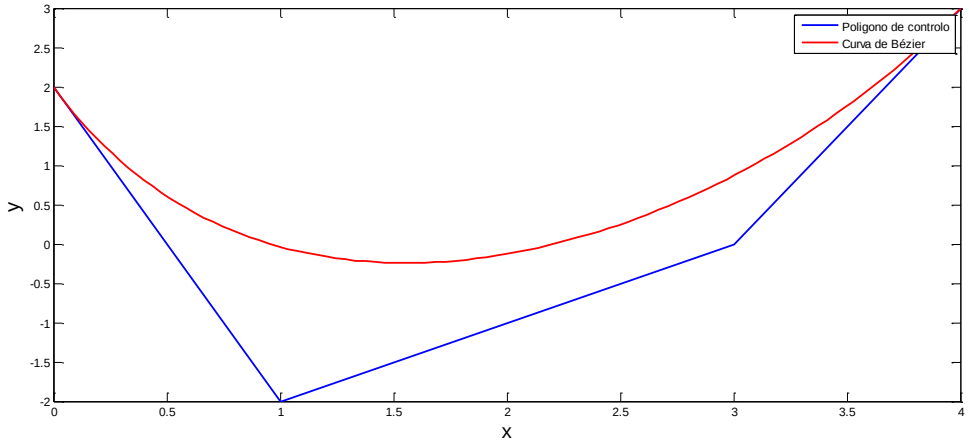


Figura 17 - Curva de Bézier do exemplo 12.

Para determinar a curva de Bézier utilizando os polinómios de Bernstein a partir de (3.26), utiliza-se a seguinte rotina ***bezier*** em Matlab:

```
%Função/ rotina para o esboço das Curvas de Bézier
%
%Desenha a curva de Bezier relativamente a um conjunto de pontos de
%controlo p = (x,y)
%
%n é o grau da curva
%
%Como saída a função apresenta as coordenadas horizontais e verticais das
%curvas de bezier, num sistema de eixos coordenados.

function [bezx,bezy]=bezier(x,y)
i=0;k=0;
n=length(x)-1;

for t=0:0.05:1,
   i=i+1;bnk=bernstein(n,k,t);ber(i)=bnk;
end;
bezx=ber*x(1);bezy=ber*y(1);
for k=1:n
   i=0;
   for s=0:0.05:1
      i=i+1;bnk=bernstein(n,k,s);ber(i)=bnk;
```

```
   end;
   bezx=bezx+ber*x(k+1);bezy=bezy+ber*y(k+1);
end;
plot(x,y,'r*')
hold on
plot(x,y,bezx,bezy)
```

Atendendo às coordenadas de P_0, P_1, P_2 *e* P_3, é possível considerar os vectores, $x=[0,1,3,4]$ e $y=[2,-2,0,3]$, que correspondem, respectivamente, às abcissas e às ordenadas dos pontos de controlo. Deste modo, invocando a rotina

```
» bezier([0,1,3,4],[2,-2,0,3])
```

obtém-se o gráfico da figura 18, no qual se pode observar o polígono de controlo constituído por segmentos de rectas que unem os pontos de controlo e a respectiva curva de Bézier.

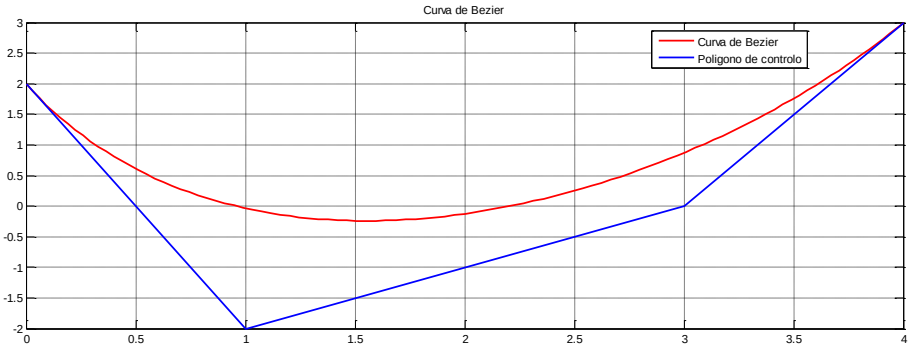


Figura 18 - Curva cúbica de Bézier do exemplo 12, obtida com a rotina ***bezier.***

3.4.2. Desvantagens das curvas de Bézier

Apesar de constituírem uma forma elegante e suave de representação dos mais variados tipos de curvas, as curvas de Bézier apresentam algumas desvantagens, tais como:

- O grau dos polinómios que definem a curva é dependente do número de pontos de controlo e, para graus elevados surgem oscilações, fazendo com que a curva não acompanhe de forma razoável o polígono de controlo;
- Curvas de grau elevado, são difíceis de desenhar e conduzem a erros de precisão consideráveis;

- Qualquer modificação em pontos de controlo provoca alterações globais na curva, tornando-se necessário recalcular toda a curva.

Ora, num sistema geométrico de construção de curvas, é desejável que, alterações em alguns pontos de uma dada curva impliquem apenas modificações nas vizinhanças desses pontos e não na sua totalidade. Abordam-se adiante as funções B-splines que apresentam algumas vantagens relacionadas com esses aspectos.

3.5. Comparação entre curvas de Bézier e Splines paramétricos

Compara-se, agora, as curvas cúbicas de Bézier com Splines cúbicos paramétricos, principalmente, no que diz respeito à facilidade de adaptação na modelação de comportamentos que apresentem elevados graus de variabilidade geométrica (William & Richard, 1974).

Para o efeito, estuda-se a aplicabilidade destas técnicas no acompanhamento do polígono de controlo dado na figura 19, que retrata os contornos de um esquilo (animal mamífero roedor).

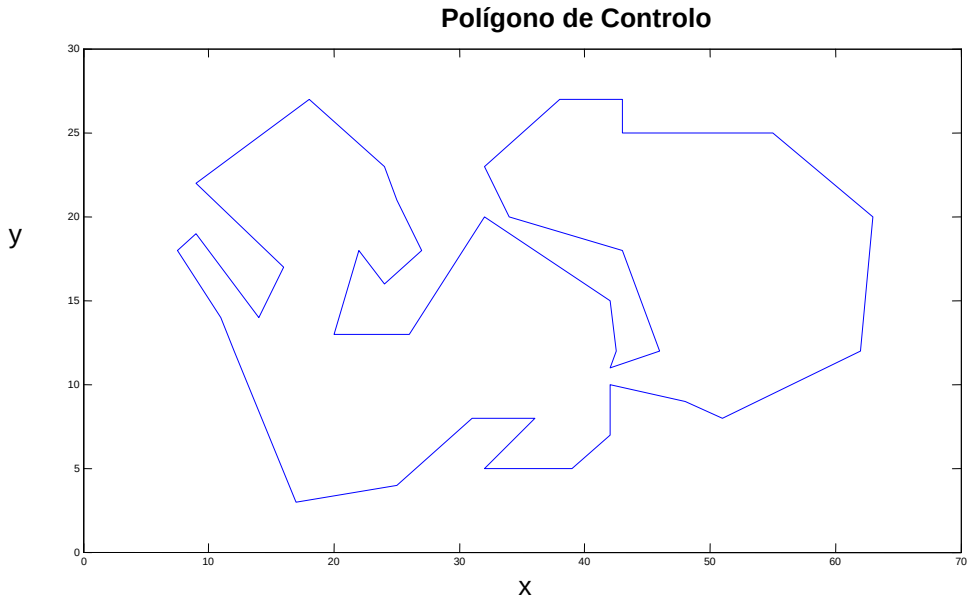


Figura 19 – Polígono de controlo a modelar por curva de Bezier e Spline paramétrico.

Assim, definem-se 40 vértices (pontos de controlo) do polígono de controlo representado e constrói-se, então, a rotina ***grafico*** em MatLab, a qual tem como entrada os pontos de controlo (vértices do polígono) e, como saída, um gráfico com a sobreposição do polígono de controlo, da curva de Bézier e do Spline paramétrico.

A rotina ***grafico*** utiliza as rotinas ***bezier*** (para a construção da curva de Bézier) e ***spline_param*** (para a construção do Spline paramétrico).

Para a utilização da rotina ***grafico***, definem-se os pontos de controlo que representam cada um dos vértices do polígono representado na figura 19.

Ou seja, neste caso, define-se

x=[7.5,11,12,17,25,31,36,32,39,42,42,48,51,62,63,55,43,43,38,32,34,43,46,42,42.5,42,32,26,20,22,24,27,25,24,18,9,16,14,9,7.5]

e

y=[18,14,12,3,4,8,8,5,5,7,10,9,8,12,20,25,25,27,27,23,20,18,12,11,12,15,20,13,13,18,16,18,21,23,27,22,17,14,19,18]

Usando a rotina ***grafico*** com

>>**grafico**([7.5,11,12,17,25,31,36,32,39,42,42,48,51,62,63,55,43,43,38,32,34,43,46,42,42.5,42,32,26,20,22,24,27,25,24,18,9,16,14,9,7.5],[18,14,12,3,4,8,8,5,5,7,10,9,8,12,20,25,25,27,27,23,20,18,12,11,12,15,20,13,13,18,16,18,21,23,27,22,17,14,19,18],39)

esta permite obter os gráficos da figura 20.

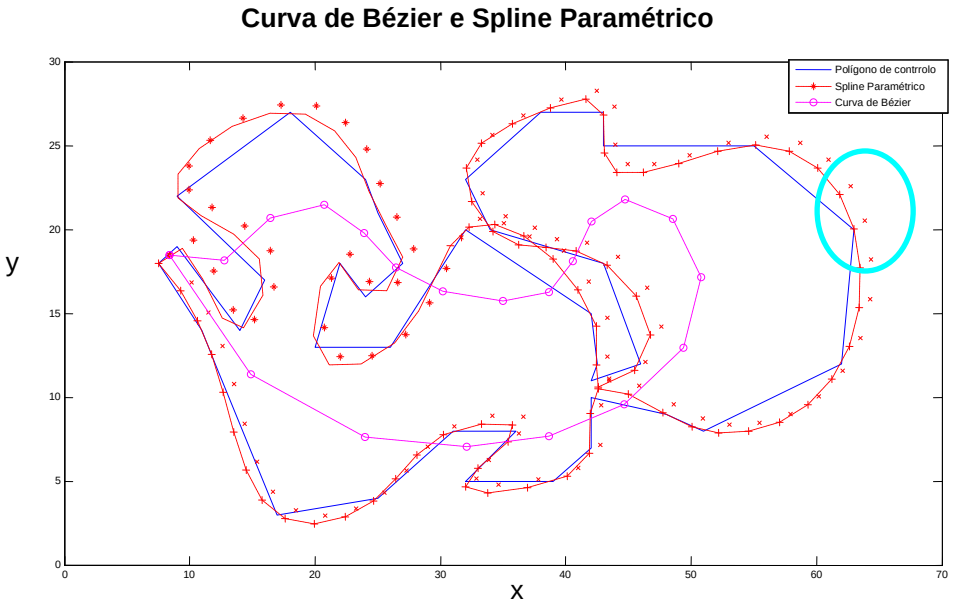


Figura 20 – Modelação comparando a curva de Bézier com o Spline paramétrico.

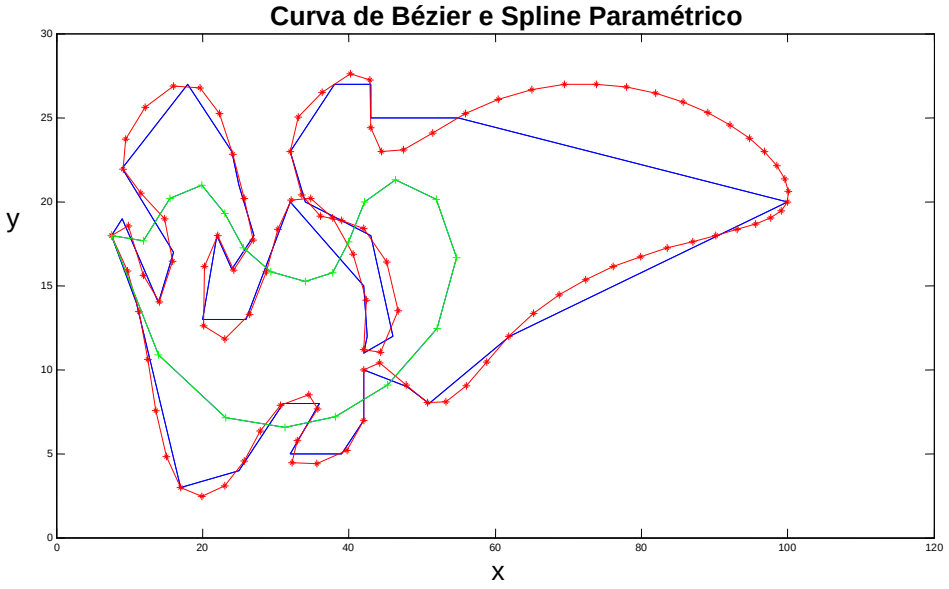


Figura 21 – Modelação comparando a curva de Bézier com o Spline paramétrico, após alteração local de um dos vértices do polígono de controlo.

Conforme pode ser observado na figura 20, o Spline paramétrico acompanha de perto os contornos do polígono de controlo, o mesmo já não acontece com a curva de Bézier.

Além disso, modificando localmente a geometria do polígono de controlo através da alteração de um dos vértices desse polígono e aplicando novamente as duas técnicas obtêm-se os gráficos da figura 21, na qual se pode constatar uma variação local do Spline paramétrico e uma variação quase total da curva de Bézier.
Assim, esta comparação mostra a vantagem dos Splines paramétricos relativamente às curvas de Bezier, quanto à flexibilidade no ajuste a modificações do comportamento geométrico de curvas.

Introduzem-se a seguir as funções B-Splines, as quais serão posteriormente comparadas com estas duas últimas técnicas de aproximação numérica frequentemente utilizadas em modelação geométrica e computação gráfica de curvas e superfícies (Farin, 1993 & 2002).

3.6. B-Splines

Pretende-se, então, implementar um método de aproximação numérica usando polinómios de baixo grau, que, para além de garantir condições de continuidade, permita um número elevado de pontos de controlo e se adapte a possíveis alterações das coordenadas desses pontos. Tal objectivo, é conseguido com as funções B-splines que utilizam uma média ponderada de polinómios de mistura, os quais vão influenciando a curva por regiões.

As funções B-Splines são usadas, principalmente, no campo da computação gráfica e foram pela primeira vez definidas, recursivamente, em 1972, por De Boor (de Boor, 1972).

Analogamente à representação das curvas de Bezier, estas curvas são representadas através de uma combinação afim dos pontos de controlo. Mais concretamente, uma função B-Spline pode ser definida como

$$s_n(t) = \sum_{i=0}^{n} P_i B_i^n(x) \tag{3.32}$$

onde P_i são os pontos de controlo e $B_i^n(x)$ as funções polinomiais (Splines) definidas por troço (ou segmento), as quais verificam certas propriedades de continuidade. A essas funções dá-se o nome de funções de mistura ou funções de base.

Cada função de mistura define a influência de um determinado ponto de controlo na geometria final da curva, sendo que, a alteração das coordenadas desse ponto de controlo provoca uma translação da curva cuja magnitude depende dos valores daquela função.

Tratam-se, a seguir, as funções B-Splines de base de grau mais baixo, zero, um e dois, a partir das quais todas as funções B-Splines podem ser obtidas.

3.6.1. B-Splines de grau zero, um e dois

As funções de base B–Spline de grau zero são denotadas por B_i^0 e apresentam o aspecto representado na figura 22.

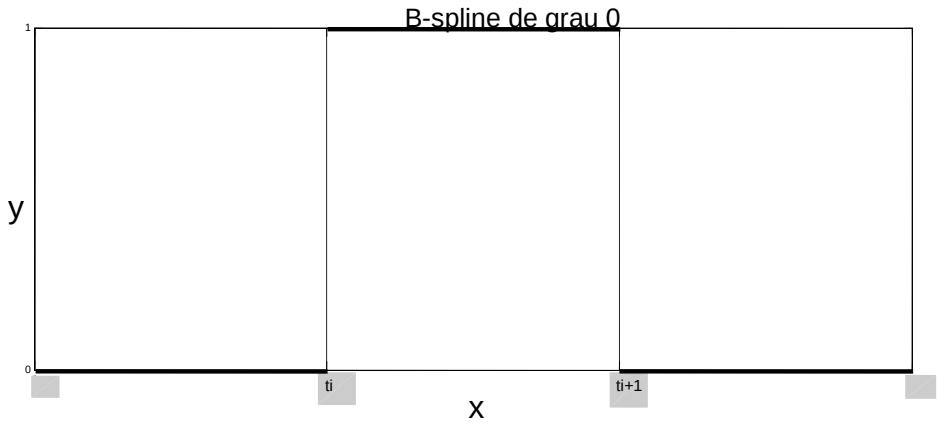


Figura 22 - B-spline de grau zero

A sua definição formal é dada por

$$B_i^0(x) = \begin{cases} 1 & se \quad x \in [t_i, t_{i+1}] \\ 0 & caso \quad contrario \end{cases} \tag{3.33}$$

e satisfazem as seguintes propriedades:

i) $B_i^0(x) \geq 0$,

ii) $\sum_{i=-\infty}^{\infty} B_i^0(x) = 1$.

As funções B_i^0 representam as bases para uma definição recursiva de todas as funções B–Spline de grau maior ou igual a 1.

Tal definição recursiva é dada por

$$B_i^k(x) = \frac{x - t_i}{t_{i+k} - t_i} B_i^{k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1}^{k-1}(x) \qquad k = 1,2,... \tag{3.34}$$

Tendo em conta a equação (3.34), podemos deduzir a fórmula para as funções de base B_i^1 da seguinte forma:

$$B_i^1(x) = \frac{x - t_i}{t_{i+1} - t_i} B_i^0(x) + \frac{t_{i+2} - x}{t_{i+2} - t_{i+1}} B_{i+1}^0(x) \tag{3.35}$$

Usando a igualdade (3.33) podemos simplificar a equação anterior, vindo

$$B_i^1(x) = \begin{cases} 0 & se\ x \le t_i\ ou\ x \ge t_{i+2} \\ \dfrac{x - t_i}{t_{i+1} - t_i} & se\ t_i \le x \le t_{i+1} \\ \dfrac{t_{i+2} - x}{t_{i+2} - t_{i+1}} & se\ t_{i+1} \le x \le t_{i+2} \end{cases} \tag{3.36}$$

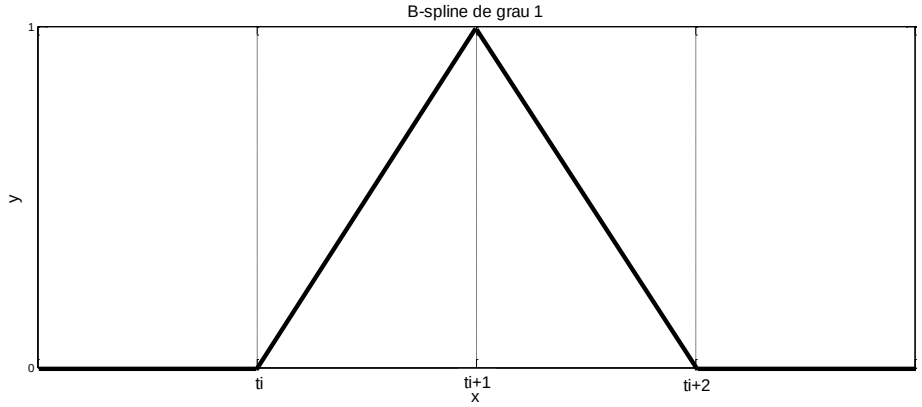


Figura 23 - B-spline de grau 1

Na figura 23 representa-se a função de base B_i^1 que apresenta as seguintes propriedades:

i) o suporte de B_i^1 é (t_i, t_{i+2});

ii) $B_i^1(x) \ge 0$;

iii) B_i^1 é continua e diferenciável em todos os pontos, com excepção dos pontos t_i, t_{i+1}, e t_{i+2};

iv) $\sum_{i=-\infty}^{\infty} B_i^1(x) = 1$.

A partir da relação recursiva (3.34), podem escrever-se as funções de base do B-Spline quadrático, obtendo-se

$$B_i^2(x) = \frac{x - t_i}{t_{i+2} - t_i} B_i^1(x) + \frac{t_{i+3} - x}{t_{i+3} - t_{i+1}} B_{i+1}^1(x), \tag{3.37}$$

onde $B_i^1(x)$ **e** $B_{i+1}^1(x)$ são as função de base B-splines lineares, definidas por (3.35) e (3.36). Deste modo, usando estas relações e, após simplificações adequadas, chega-se à seguinte definição para as funções de base dos B-splines quadráticos:

$$B_i^2(x) = \begin{cases} 0, & x \le t_i \quad ou \quad x \ge t_{i+3} \\ \dfrac{(x - t_i)^2}{(t_{i+1} - t_i)(t_{i+2} - t_i)}, & t_i \le x \le t_{i+1} \\ \dfrac{(x - t_i)(t_{i+2} - x)}{(t_{i+2} - t_i)(t_{i+2} - t_{i+1})} + \dfrac{(t_{i+3} - x)(x - t_{i+1})}{(t_{i+3} - t_{i+1})(t_{i+2} - t_{i+1})}, & t_{i+1} \le x \le t_{i+2} \\ \dfrac{(t_{i+3} - x)^2}{(t_{i+3} - t_{i+1})(t_{i+3} - t_{i+2})}, & t_{i+2} \le x \le t_{i+3} \end{cases} \tag{3.38}$$

Na figura 24 pode observar-se o exemplo de uma função de base B-spline quadrática.

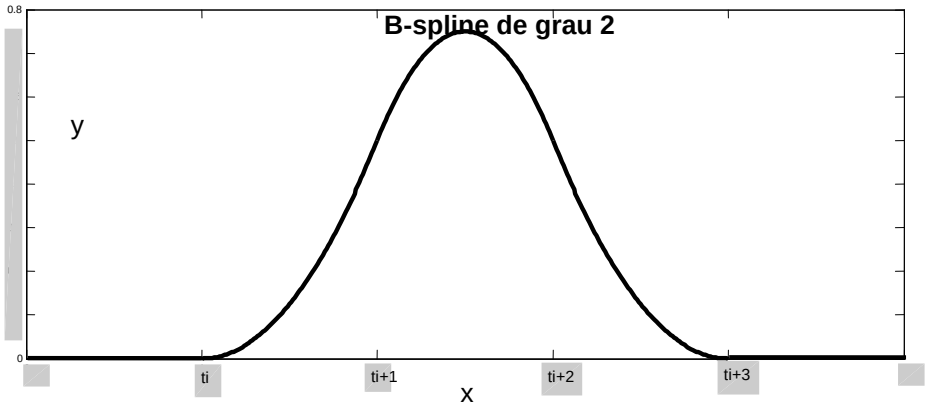


Figura 24 - B-spline quadrático

3.6.2. B-Splines cúbicos

As funçãos de base do B-spline cúbico são obtidas, recursivamente, a partir da expressão (3.34), sendo dadas por

$$B_i^3(x) = \frac{x - t_i}{t_{i+3} - t_i} B_i^2(x) + \frac{t_{i+4} - x}{t_{i+4} - t_{i+2}} B_{i+1}^2(x) \tag{3.39}$$

onde $B_i^2(x)$ e $B_{i+1}^2(x)$ são as funções de base dos B-splines quadráticos.

Usando (3.37) e (3.38), após simplificações, obtém-se:

$$B_i^3(x) = \begin{cases} 0, & x \le t_i \quad ou \quad x \ge t_{i+4} \\ \dfrac{(x - t_i)^3}{(t_{i+1} - t_i)(t_{i+2} - t_i)(t_{i+3} - t_i)}, & t_i \le x \le t_{i+1} \\ \dfrac{x - t_i}{t_{i+3} - t_i}\left[\dfrac{(x - t_i)(t_{i+2} - x)}{(t_{i+2} - t_i)(t_{i+2} - t_{i+1})} + \dfrac{(t_{i+3} - x)(x - t_{i+1})}{(t_{i+3} - t_{i+1})(t_{i+2} - t_{i+1})}\right] + \dfrac{t_{i+4} - x}{t_{i+4} - t_{i+2}} \dfrac{(x - t_{i+1})^2}{(t_{i+2} - t_{i+1})(t_{i+3} - t_{i+1})}, & t_{i+1} \le x \le t_{i+2} \\ \dfrac{x - t_i}{t_{i+3} - t_i} \dfrac{(t_{i+3} - x)^2}{(t_{i+3} - t_{i+1})(t_{i+3} - t_{i+2})} + \dfrac{t_{i+4} - x}{t_{i+4} - t_{i+2}}\left[\dfrac{(x - t_{i+1})(t_{i+3} - x)}{(t_{i+3} - t_{i+1})(t_{i+3} - t_{i+2})} + \dfrac{(t_{i+4} - x)(x - t_{i+2})}{(t_{i+4} - t_{i+2})(t_{i+3} - t_{i-2})}\right], & t_{i+2} \le x \le t_{i+3} \\ \dfrac{(t_{i+4} - x)^3}{(t_{i+3} - t_i)(t_{i+3} - t_{i+1})(t_{i+3} - t_{i+2})}, & t_{i+3} \le x \le t_{i+4} \end{cases}$$

Na figura 25 faz-se uma representação gráfica das funções de base B-splines cúbicas.

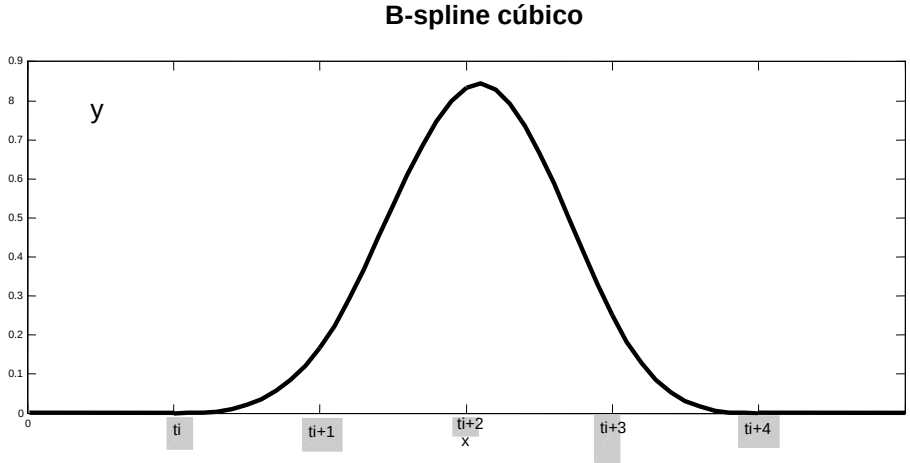


Figura 25 - B-spline cúbico

Ilustra-se, agora, uma aplicação prática dos B-splines cúbicos na modelação geométrica da curvatura associada ao design de um veículo (carrinha pick up).

Para o efeito, define-se o polígono de controlo da figura 26 que, de uma forma grosseira, representa o contorno linear do veículo.

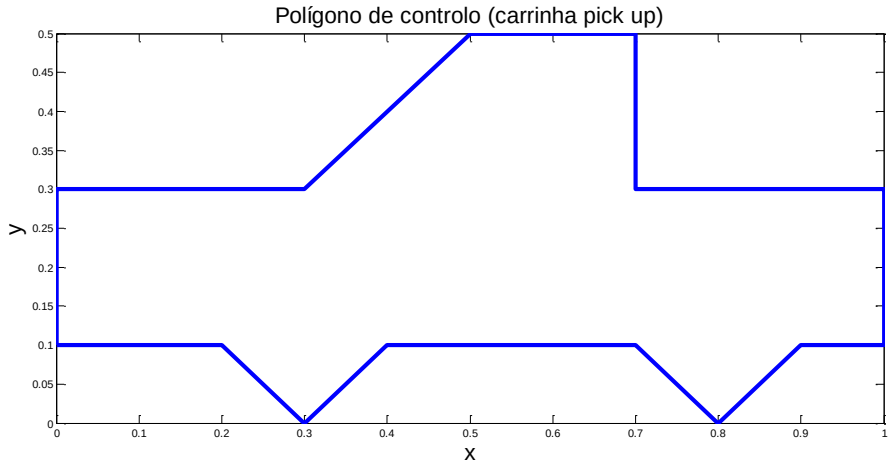


Figura 26 – Polígono de controlo de um veículo (carrinha pick up).

Utilizando a rotina ***bspline*** apresentada na secção seguinte e tomando 15 pontos de controlo, obtém-se a curva B-Spline cúbica da figura 27, a qual já apresenta caracteristicas suaves de continuidade, mas que pode ser melhorada aumentando o número de pontos de controlo.

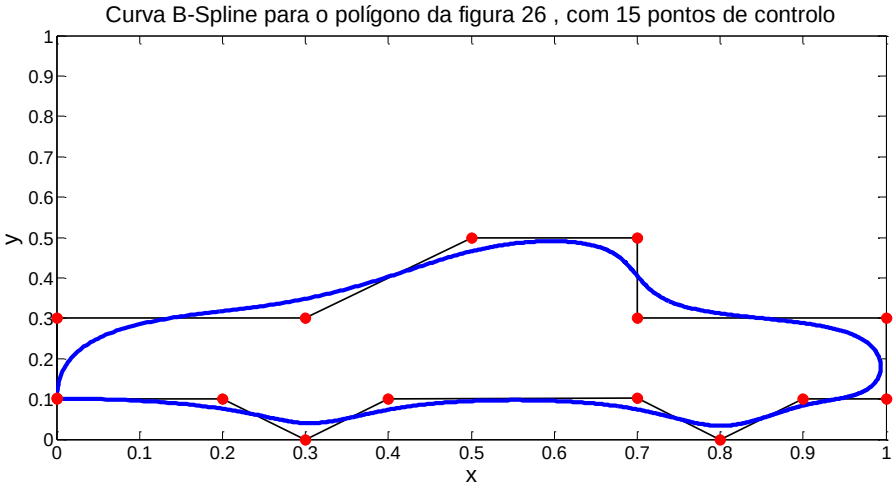


Figura 27 – Curva B-Spline cúbica com 15 pontos de controlo para modelação do polígono da figura 26.

Deste modo, na figura 28, pode ver-se que com 31 pontos de controlo, se consegue uma razoável aproximação na modelação da curvatura associada ao design do veículo.

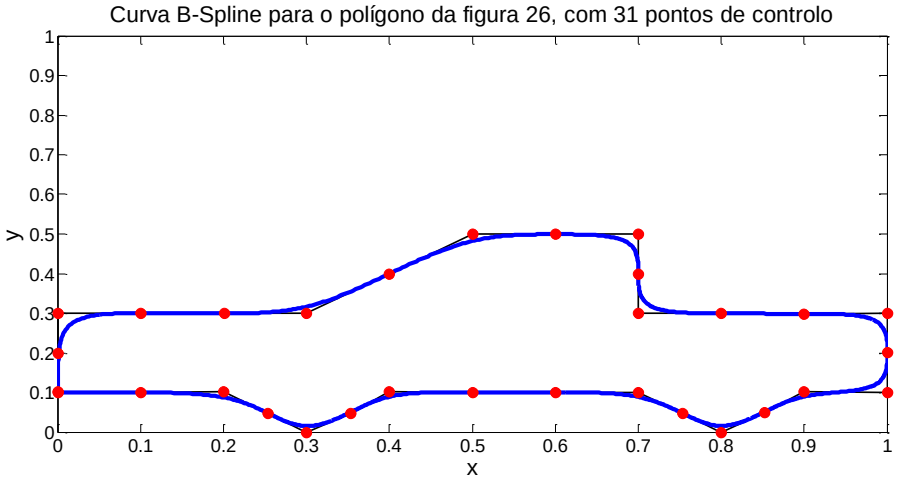


Figura 28 – Curva B-Spline cúbica com 31 pontos de controlo para modelação do polígono da figura 26.

3.7. Comparação entre curvas de Bézier, B-Splines e Splines paramétricos

Para comparar as curvas de Bezier, B-Splines e Splines paramétricos, todos de grau três, considere-se o polígono dado na figura 29, o qual inclui 12 pontos de controlo.

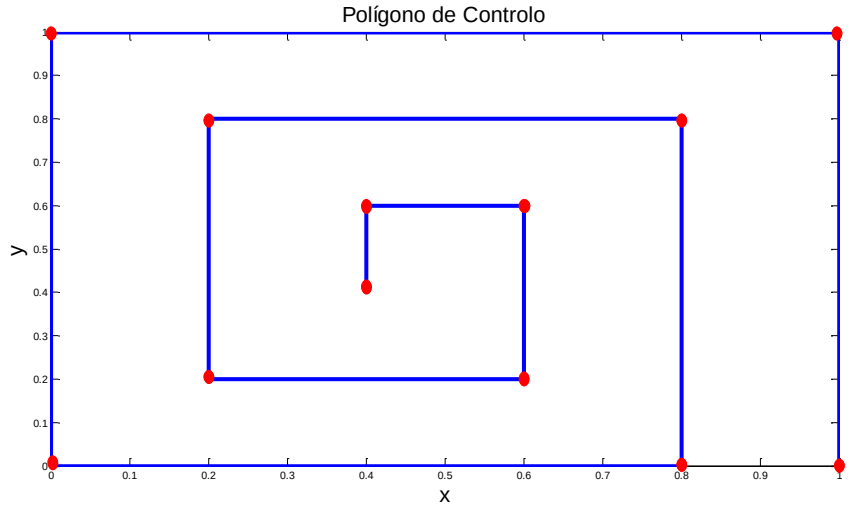


Figura 29 – Polígono com 12 pontos de controlo para comparação das curvas de Bézier, B-Splines e Splines paramétricos.

A obtenção das curvas para esta análise comparativa é feita através de simulações numéricas com as rotinas ***bezier,*** introduzida na secção 3.4.1., e ***bspline***, a seguir apresentada, que foi desenvolvida por Stefan Hüeber da Universidade de Stuttgart, em 2007. A rotina ***bspline*** utiliza a rotina ***DEBOOR***, que é dada logo a seguir e cuja versão aqui apresentada foi desenvolvida por Jonas Ballani em 2007.

```
function bspline(n,order)

% function spline(n,order)
%
% Plots the B-slpine-curve of n control-points.
% The control points can be chosen by clicking
% with the mouse on the figure.
%
% COMMAND:  spline(n,order)
% INPUT:    n     Number of Control-Points
%           order Order ob B-Splines
%                 Argnument is arbitrary
%                 default: order = 4
%
% Date:     2007-11-28
% Author:   Stefan Hüeber

close all;
if (nargin ~= 2)
    order = 4;
end
nplot = 100;

if (n < order)
    display([' !!! Error: Choose n >= order=',num2str(order),' !!!']);
    return;
end

figure(1);
axis([0 1 0 1]);
hold on; box on;
set(gca,'Fontsize',16);

t = linspace(0,1,nplot);

for i = 1:n
    title(['Choose ',num2str(i),' th. control point']);
    p(i,:) = ginput(1);
    hold off;
    plot(p(:,1),p(:,2),'k-','LineWidth',2);
    axis([0 1 0 1]);
    hold on; box on;
    if (i  >= order)
        T = linspace(0,1,i-order+2);
        y = linspace(0,1,1000);
        p_spl = DEBOOR(T,p,y,order);
        plot(p_spl(:,1),p_spl(:,2),'b-','LineWidth',4);
```

```
    end
    plot(p(:,1),p(:,2),'ro','MarkerSize',10,'MarkerFaceColor','r');
end

title(['B-Spline-curve with ',num2str(n),' control points of order
',num2str(order)]);


function val = DEBOOR(T,p,y,order)

% function val = DEBOOR(T,p,y,order)
%
% INPUT:  T     Stützstellen
%         p     Kontrollpunkte (nx2-Matrix)
%         y     Auswertungspunkte (Spaltenvektor)
%         order Spline-Ordnung
%
% OUTPUT: val   Werte des B-Splines an y (mx2-Matrix)
%
% Date:   2007-11-27
% Author: Jonas Ballani
p
m = size(p,1);
n = length(y)
X = zeros(order,order);
Y = zeros(order,order);
a = T(1);
b = T(end);
T = [ones(1,order-1)*a,T,ones(1,order-1)*b];


for l = 1:n
    t0 = y(l);
    id = find(t0 >= T);
    k = id(end);
        if (k > m)
            return;
        end
    X(:,1) = p(k-order+1:k,1);
    Y(:,1) = p(k-order+1:k,2);

    for i = 2:order
        for j = i:order
            num = t0-T(k-order+j);
            if num == 0
                weight = 0;
            else
                                    s = T(k+j-i+1)-T(k-order+j);
                weight = num/s;
            end
            X(j,i) = (1-weight)*X(j-1,i-1) + weight*X(j,i-1);
            Y(j,i) = (1-weight)*Y(j-1,i-1) + weight*Y(j,i-1);
        end
    end
    val(l,1) = X(order,order);
    val(l,2) = Y(order,order);
end
```

São utilizados 12 pontos de controlo, pela ordem seguinte:
(1,0), (1,1), (0,1), (0,0), (0.8,0), (0.8,0.8), (0.2,0.8), (0.2,0.2), (0.6,0.2), (0.6,0.6), (0.4,0.4) e (0.4,0.2).

Assim, obtêm-se os gráficos da figura 30. De seguida, procede-se a uma alteração de um vértice do polígono de controlo, resultando, pelo mesmo processo de simulação numérica, os gráficos da figura 31.

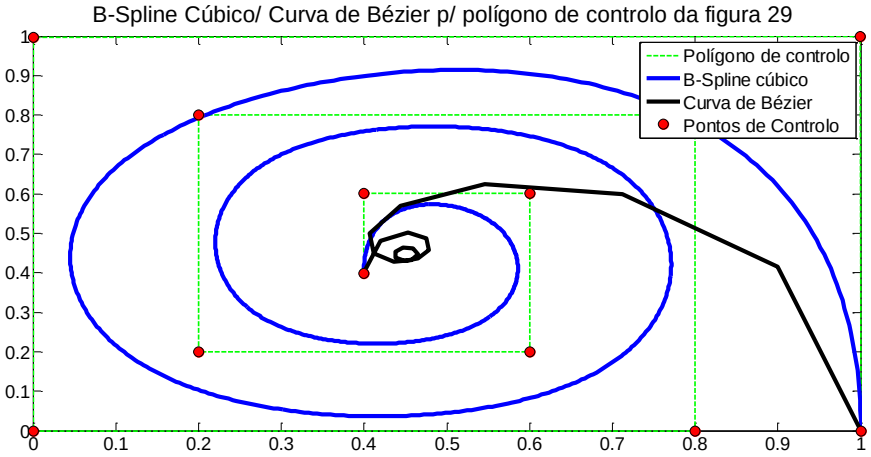


Figura 30 – Comparação do B-Spline com a curva de Bézier na modelação do polígono da figura 29.

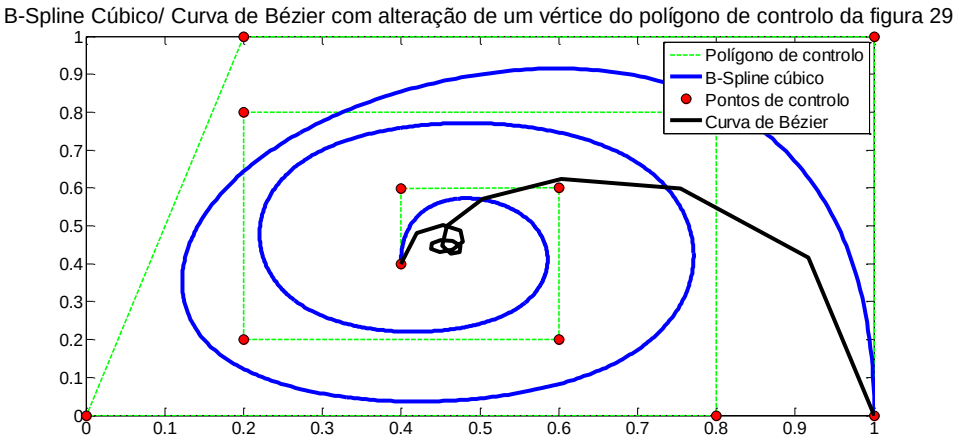


Figura 31 – Comparação do B-Spline com a curva de Bézier na modelação do polígono da figura 29, após alteração local de um vértice desse polígono.

Observando as figuras 30 e 31, verifica-se uma grande distorsão nas curvas de Bezier motivada pela variabilidade do traçado do polígono (em forma de espiral), sendo que, nestes casos, não é aconselhável a utilização deste tipo de curvas para modelação geométrica. Além disso, é possível identificar as seguintes vantagens das funções B-Splines:

- A perturbação de um único vértice do polígono de controlo produz uma perturbação local da curva, nas proximidades do vértice perturbado;
- As curvas de funções B-Splines aproximam o polígono de controlo de uma forma mais próxima, ou seja, acompanham melhor a variabilidade geométrica deste.

De modo análogo se procede com o mesmo polígono de controlo da figura 29, para efectuar uma comparação entre a curva B-Spline e o Spline paramétrico, obtendo-se os gráficos representados na figura 32.

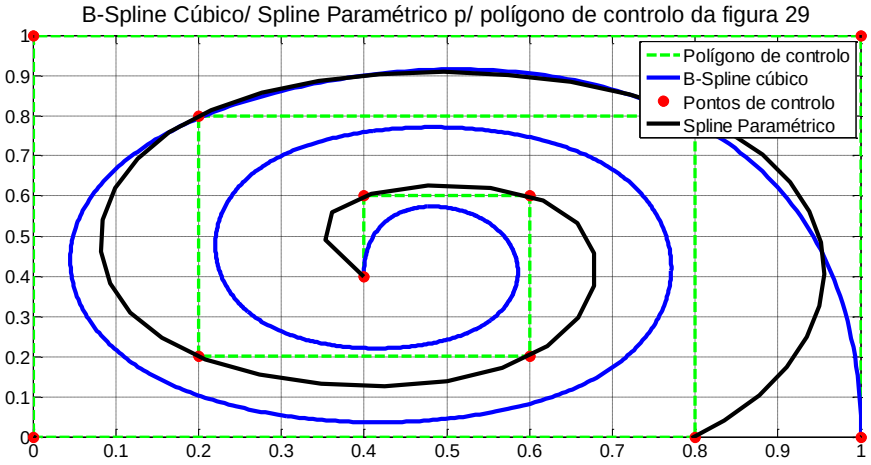


Figura 32 – Comparação do B-Spline com o Spline paramétrico
na modelação do polígono da figura 29.

Analisando os gráficos obtidos nesta figura, pode dizer-se que estes dois métodos contornam as dificuldades inerentes às curvas de Bezier, apresentando ambos bons resultados, sendo que, neste caso, o Spline paramétrico consegue melhorar o seguimento da variabilidade do polígono de controlo nalgumas regiões, quando comparado com o comportamento da curva associada ao B-Spline.

4. Interpolação Bidimensional

4.1. Introdução

Neste capítulo, faz-se uma extensão da interpolação polinomial ao caso bidimensional. Deste modo, parte-se do princípio de que a função $f(x,y)$ é conhecida num conjunto de pontos $(x_i, y_j), i = 0,1,...,n \quad e \quad j = 0,1,...,m$.

Para o efeito, denota-se por Ω um subconjunto do domínio R^2 e, por $\bar{x} = (x,y)$ a coordenada vectorial de um ponto de Ω.

A situação mais fácil ocorre quando $\Omega = [a,b] \times [c,d]$. Nesse caso, considerando as partições unidimensionais, $\Delta x: \; a = x_0 < x_1 < x_2 < ... < x_n = b$ e $\Delta y: \; c = y_0 < y_1 < y_2 < ... < y_m = d$, é possível definir uma partição rectangular de Ω, como sendo o produto tensorial de duas partições unidimensionais, ou seja, $\Omega: \; \Delta = \Delta x \times \Delta y$, que consiste em $n \cdot m$ subrectângulos, $\Omega_{ij} = [x_{i-1}, x_i] \times [y_{i-1}, y_i], \; (i = 1,2,...,n; j = 1,2,...,m)$.

Cada um desses subrectângulos terá como lados os designados parâmetros de malha

$h_i = x_i - x_{i-1}, \; i = 1,2,...,n$ e $g_j = y_j - y_{j-1}, \; j = 1,2,...,m$.

Muitas vezes, toma-se como parâmetro de malha $h = h_i = g_j$.

As linhas que unem os pontos $x_i, \; i = 0,1,..,n$ e as que unem os pontos $y_j, \; j = 0,1,..,m$ são designadas de linhas da grelha da partição Δ. Os pontos de intersecção de linhas da grelha, $(x_i, y_j) \; (i = 0,1,...,n; j = 0,1,...,m)$, são designados de nós da partição Δ.

Como é evidente, existem $(m+1)*(n+1)$ nós da partição Δ, como mostra a figura 33.

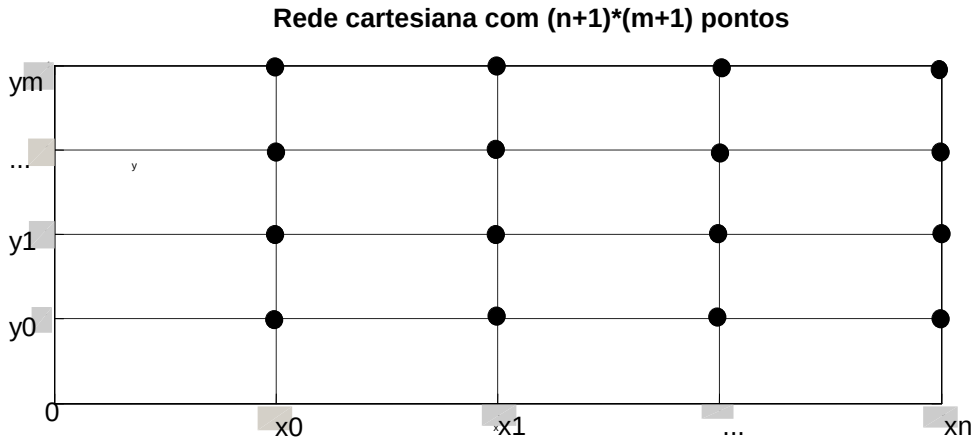


Figura 33 – Rede cartesiana para interpolação bidimensional com $(m+1)*(n+1)$ nós.

4.2. Forma de Lagrange bidimensional

Neste caso, o polinómio interpolador na forma de Lagrange pode ser escrito como

$$P_{n,m} = P_{n,m}(x,y) = \sum_{i=0}^{n}\sum_{j=0}^{m} \alpha_{ij} u_i(x) v_j(y) \qquad (4.1)$$

Onde $u_i(x) \in P_n$, $i = 0,1,...,n$ e $v_j(y) \in P_m$, $j = 0,1,...,m$, são os polinómios fundamentais de Lagrange unidimensionais, relativos a x e a y, respectivamente, os quais podem ser obtidos usando, adequadamente, a expressão (2.6) e $\alpha_{ij} = f(x_i, y_j)$, onde $f(x,y)$ é a função bidimensional a interpolar.

Exemplo 13

Determinar o polinómio interpolador $P_{n,m}$, de duas variáveis, que interpola a função $f(x,y) = \mathbf{sin}(x^2 + y^2)$ nos pontos, $x = -1,0,1 \quad e \quad y = -1,0,1$.

Na figura 34 representa-se o gráfico da função $f(x,y) = \mathbf{sin}(x^2 + y^2)$, obtido através da seguinte sequência de comandos em MatLab, com parâmetro de malha h=0.05:

```
[x, y] = meshgrid(-1:.05:1);
z = sin(x.^2 + y.^2);
surf(x, y, z), title('Gráfico da função sin(x^2 + y^2)')
```

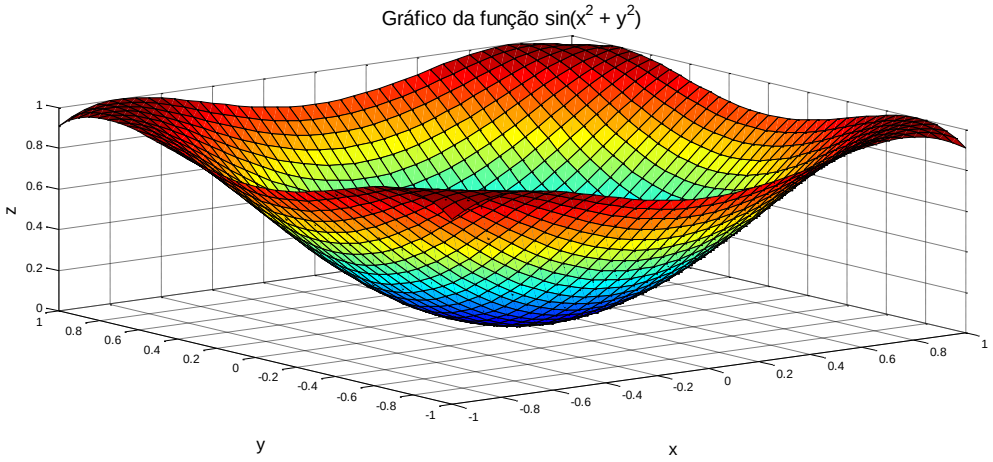
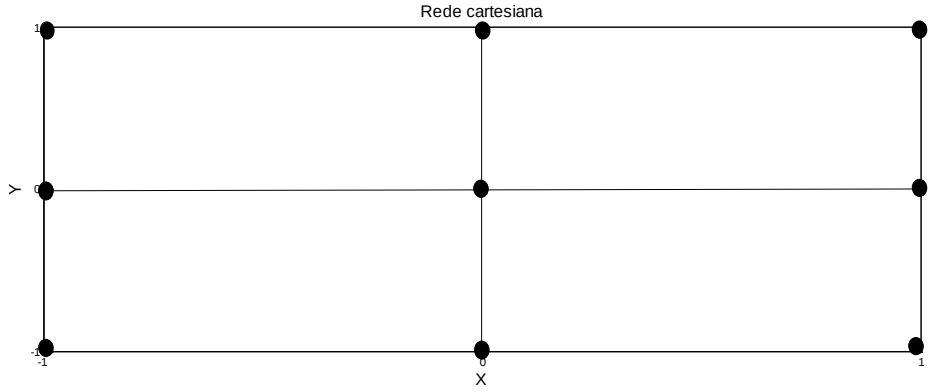


Figura 34 - Gráfico da função f(x,y)=sin(x^2+y^2).

Com os pontos dados, é possível definir os nós da grelha através da seguinte tabela

(x,y)	(-1,-1)	(-1,0)	(-1,1)	(0,-1)	(0,0)	(0,1)	(1,-1)	(1,0)	(1,1)
f(x,y)	0.9093	0.8415	0.9093	0.8415	0	0.8415	0.9093	0.8415	0.9093

cuja rede cartesiana se pode representar como



Da expressão (2.6), podem obter-se os polinómios fundamentais de lagrange, relativos a x e a y, vindo

$$u_0(x) = \frac{(x-0)(x-1)}{(-1-0)(-1-1)} = \frac{1}{2}x(x-1)$$

$$u_1(x) = \frac{(x+1)(x-1)}{(0+1)(0-1)} = -(x+1)(x-1)$$

$$u_2(x)=\frac{(x+1)(x-0)}{(1+1)(1-0)}=\frac{1}{2}(x+1)x$$

e

$$v_0(x)=\frac{(y-0)(y-1)}{(-1-0)(-1-1)}=\frac{1}{2}y(y-1)$$

$$v_1(x)=\frac{(y+1)(y-1)}{(0+1)(0-1)}=-(y+1)(y-1)$$

$$v_2(x)=\frac{(y+1)(y-0)}{(1+1)(1-0)}=\frac{1}{2}(y+1)y$$

Donde, utilizando (4.1), resulta o polinómio interpolador bidimensional $P_{2,2}$

$$\begin{aligned}P_{2,2}=u_0(x)[&f(x_0,y_0)v_0(y)+f(x_0,y_1)v_1(y)+f(x_0,y_2)v_2(y)]\\&+u_1(x)[f(x_1,y_0)v_0(y)+f(x_1,y_1)v_1(y)+f(x_1,y_2)v_2(y)]\\&+u_2(x)[f(x_2,y_0)v_0(y)+f(x_2,y_1)v_1(y)+f(x_2,y_2)v_2(y)]\end{aligned}$$

o qual, na forma simplificada, pode ser escrito como

$$P_{2,2}=-0.77365x^2y^2+0.8415x^2+0.00005x^2y+0.84145y^2-0.00005y\,.$$

Este polinómio está representado na figura 35, tendo-se usado para o obter a sequência seguinte de comandos em MatLab, com parâmetro de malha h=0.05:

```
[x,y]=meshgrid(-1:0.05:1);
b=x.^2;
a=y.^2;
z=(-0.77365.*b.*a)+(0.8415.*b)+(0.00005.*b.*y)+(0.84145.*a)-(0.00005.*y);
surf(x, y, z), title('Gráfico da Polinómio P2,2')
```

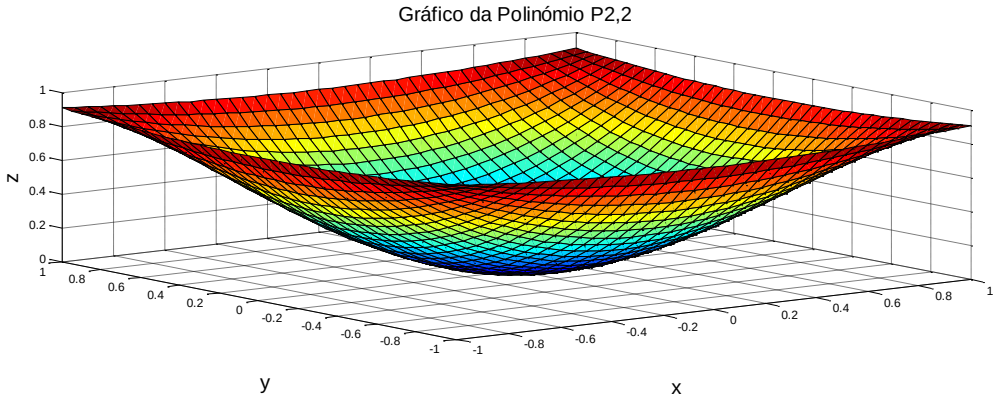


Figura 35 - Gráfico do polinomio P2,2, interpolador da função f(x,y)=sin(x^2+y^2).

A seguir, faz-se uma representação gráfica dos polinómios interpoladores, dependentes de duas variáveis, x e y, utilizando interpolação bidimensional linear e cúbica.

De salientar que, no caso linear, o valor de um determinado ponto interpolado é obtido pela combinação dos valores dos quatro pontos mais próximos, enquanto que, no caso da interpolação cúbica, o valor de um ponto interpolado é obtido pela combinação dos valores dos dezasseis pontos mais próximos. Como pode ser constatado através das figuras 36 e 37, a interpolação bidimensional cúbica (bi-cúbica) fornece uma superfície mais suave do que a obtida com a interpolação bidimensional linear (bi-linear).

Para obter os respectivos polinómios interpoladores bidimensionais, começa-se por definir a malha, neste caso, utiliza-se $-1 \le x \le 1$, $-1 \le y \le 1$ e procede-se da seguinte forma, usando rotinas do MatLab adequadas:

```
[x, y] = meshgrid(-1:.05:1);
z = sin(x.^2 + y.^2);
[xi, yi] = meshgrid(-1:.05:1);
zi = interp2(x, y, z, xi, yi, 'linear');
surf(xi,yi,zi),title('Interpolação bi-linear da função sin(x^2 + y^2)')
```

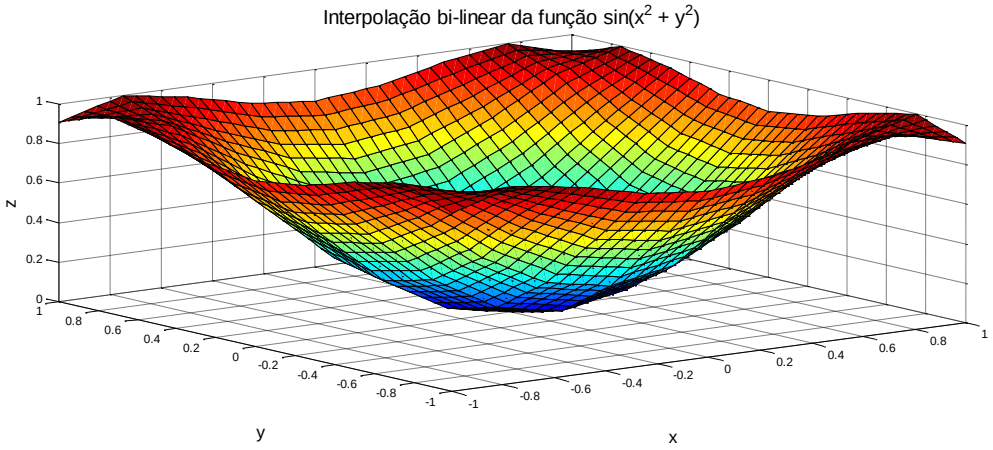


Figura 36 - Interpolação bi-linear da função f(x,y)=sin(x^2+y^2) com h=0.05.

```
[x, y] = meshgrid(-1:.05:1);
z = sin(x.^2 + y.^2);
[xi, yi] = meshgrid(-1:.05:1);
zi = interp2(x, y, z, xi, yi, 'cubic');
surf(xi,yi,zi),title('Interpolação bi-cúbica da função sin(x^2 + y^2)')
```

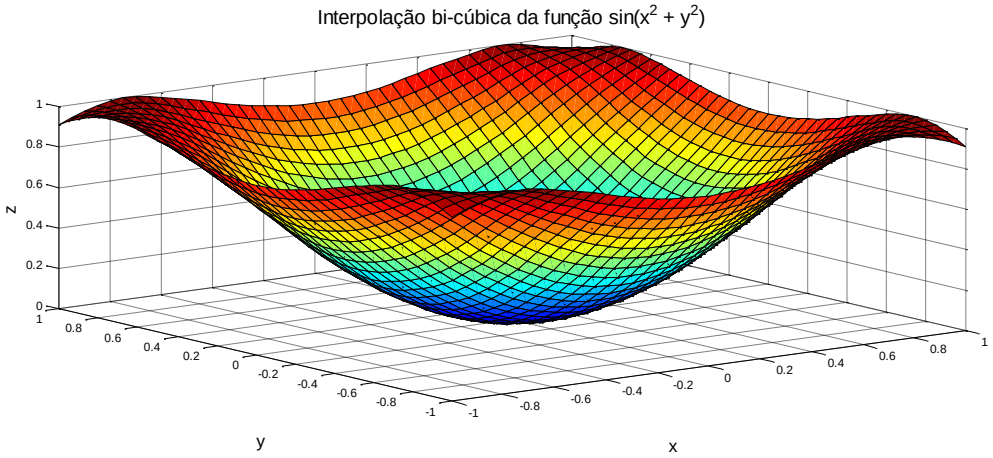


Figura 37 - Interpolação bi-cúbica da função f(x,y)=sin(x^2+y^2 com h=0.05.

Conforme se pode verificar pela figura 37, a interpolação bi-cúbica com um parâmetro de malha, h=0.05, reduzido, consegue uma razoável aproximação da função interpolada $f(x,y)=\mathbf{sin}(x^2+y^2)$.

Para malhas de parâmetro elevado (figura 38), o erro na aproximação aumenta consideravelmente.

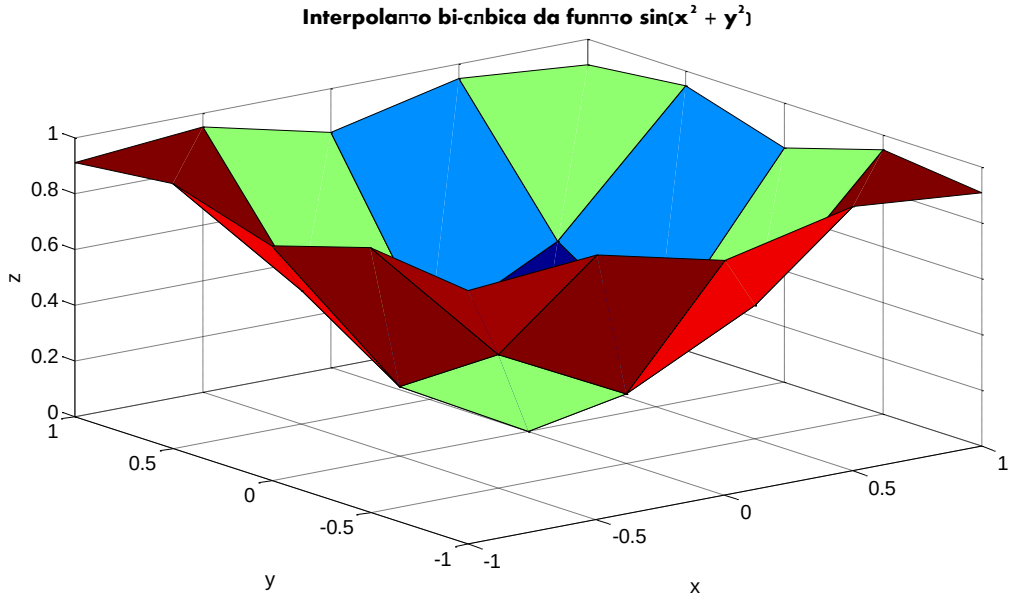


Figura 38 - Interpolação bi-cúbica da f(x,y)=sin(x^2+y^2 com h=0.5.

Conclusão

Neste trabalho abordaram-se alguns métodos de aproximação numérica, tirando partido da ferramenta computacional Matlab no desenvolvimento de rotinas que possibilitaram investigar a aplicabilidade desses métodos e tirar conclusões quanto às suas vantagens/desvantagens.

De uma forma geral, conhecidos os valores de uma determinada função f num conjunto de nós, $x_0, x_1, ..., x_n$, torna-se possível construir um polinómio interpolador de f naquele suporte de nós dados. Tal polinómio pode ser construído na forma de Lagrange ou na forma de Newton.
Contudo, o polinómio na forma de Lagrange, para além de requerer um número elevado de operações, está intimamente ligado aos nós, pelo que a alteração do número ou da posição destes, conduz a uma alteração completa do polinómio. Deste modo, é preferível escrever o polinómio na forma de Newton em detrimento da forma de Lagrange.

Por outro lado, conhecidos os valores de uma determinada função f, bem como os valores da sua derivada, num conjunto de nós, $x_0, x_1, ..., x_n$, torna-se possível construir o polinómio interpolador de Hermite. Esse polinómio, para além de interpolar os valores da função, também interpola os valores da derivada da função.

Pretende-se que o polinómio interpolador esteja "muito" próximo da função interpolada, diminuindo-se deste modo o erro de interpolação cometido. Como se verificou, não é garantido que esse objectivo seja atingido com o aumento do grau do polinómio interpolador (ou seja, aumentando o número de nós de interpolação). Para conseguir tal objectivo, devem utilizar-se como nós de interpolação os zeros de um polinómio de Chebyshev de grau adequado.

Por outro lado, constatou-se que uma forma eficaz de garantir uma diminuição considerável do erro de interpolação, é dividir o intervalo de interpolação em vários subintervalos e utilizar um polinómio interpolador de grau adequado em cada um desses

subintervalos. Esta técnica é conhecida por interpolação segmentada ou interpolação com Splines, sendo mais comum recorrer-se a polinómios interpoladores de grau três naqueles subintervalos (ou segmentos), originando os chamados Splines cúbicos. Neste caso, é necessário garantirem-se propriedades de regularidade (continuidade) até às derivadas de ordem dois nos pontos intermédios que ligam os vários subintervalos, para, assim, se obter uma curva cúbica suave (Spline cúbico).

Em certas aplicações, como por exemplo, na indústria automóvel, aeronáutica e naval, onde se pretendem projectar protótipos com formas suaves, torna-se necessário recorrer a curvas paramétricas, pelo que, mostrou-se como tal é possível, definindo o comportamento da curva através de uma equação para cada um dos eixos do sistema cartesiano. Seguindo essa ideia, apresentou-se uma rotina em MatLab para simulação de Splines paramétricos.

Abordou-se também uma outra técnica de aproximação numérica muito usada em modelação geométrica e em métodos de computação gráfica, as curvas de Bézier, desenvolvidas por Pierre Bézier, engenheiro da Renault, em 1972. Mostrou-se como as curvas de Bézier podem ser obtidas utilizando o algoritmo de Casteljau, o qual, a partir de um conjunto de pontos de controlo, que são os vértices de um polígono que serve de base à modelação geométrica que se pretende construir, conduz ao gráfico da curva de Bézier. O referido algoritmo consiste em sucessivas interpolações lineares daqueles pontos de controlo, tendo sido implementado em MatLab e vistos alguns exemplos da sua aplicação.

Fez-se, ainda, uma comparação entre as curvas de Bézier e os Splines paramétricos, constatando-se as seguintes vantagens destes em relação aquelas curvas:

- O Spline paramétrico consegue um acompanhamento muito razoável do polígono de controlo definido pelos pontos de controlo;
- A alteração das coordenadas de um dado vértice do polígono de controlo modifica o comportamento global da curva de Bézier, enquanto que no caso do Spline paramétrico este sofre apenas uma variação local.

Além disso, apesar da sua forma suave e elegante de representação de curvas, as curvas de Bézier, apresentam, ainda, desvantagem quando a forma do polígono de controlo assume saliências acentuadas.

Como forma de contornar estas desvantagens são utilizadas as funções B-splines que foram definidas pela primeira vez em 1972 por Deboor, sendo aplicadas em subintervalos adequados resultantes da subdivisão do intervalo de interpolação, no qual se pretende fazer a aproximação. Estas funções B-splines, como se viu, para além de garantirem uma boa aproximação do polígono de controlo, apresentam uma grande vantagem em relação às curvas de Bézier que consiste no facto de a alteração das coordenadas de um determinado ponto de controlo, provocar somente uma alteração local da curva de aproximação numérica que se constrói, tal como acontece no caso dos Splines paramétricos.

Análises comparativas entre curvas de Bézier, Splines paramétricos e B-Splines foram feitas recorrendo a exemplos sugestivos de aplicações práticas, acompanhados pelas respectivas simulações numéricas realizadas com rotinas adequadas em MatLab.

Por fim, fez-se uma abordagem sumária à extensão da interpolação polinomial ao caso bidimensional, para interpolação de funções a duas variáveis z=f(x,y), mostrando-se como se pode construir um polinómio interpolador de Lagrange nesta situação, sendo também apresentados exemplos de experimentação numérica em MatLab.

Donde, quando se pretenda construir uma aproximação numérica conveniente para aproximar um conjunto de dados (caso discreto) ou para substituir uma função mais complicada por um polinómio aproximante mais simples (caso contínuo), pode, a partir daqui, ter-se uma noção mais concreta de um método expedito a utilizar na construção daquela aproximação. Daí, a contribuição desta dissertação no âmbito dos métodos de aproximação numérica usando o MatLab como ferramenta computacional para simulação e experimentação numérica dos mesmos.

Bibliografia

Bezier, P. (1986). *The Mathematical Basis of the UNISURF CAD System*. Butterworth-Heinemann.

Boor, C. (1972). *On calculating with B-spline.* Journal of Aproximation Theory.

Cunha, R. (2001). *Análise Numérica.* Lisboa, ISEL edições.

Dahlquist, G. & Bjorck, A. (2003). *Numerical Methods*. Dover Publications, Inc. Mineola, New York.

Deufland, P. & Hohmann, A. (2003). *Numerical Analysis in modern scientific computing – an introduction*. Second Editon, Springer.

Farin, G. (1992). *Curves and Surfaces for Computer Aided Geometric Design (CAGD).*

Farin, G. (2002). *Curves and Surfaces for CAGD (Fifth Edition) - A Practical Guide***.** Elsevier Inc.

Fausett, L. V. (1999). *Applied Numerical Analysis Using MATLAB.* Prentice Hall, Upper Saddle River, NJ.

George, C. (2003). *Fundamental Numerical Methods and Data Analysis.* George Collins.

Hunes, A. F. P. e Melo, I. S. H. (1983). *Noções de Cálculo Numérico*. McGraw-Hill Editora, São Paulo.

Iyengar, S. & Jain, R. (2009). *Numerical Methods.* New age international Limited Publishers.

Karl, G. S. (2006). *The history of approximation theor.* Birkhauser, Boston.

Karris, S. (2004). *Numerical Analysis using MATLAB and Spreadsheets*. Second Edition, Orchard Publications.

Kincaid, D. & Cheney, W. (2002). *Numerical Analysis: Mathematics of Scientific Computing*. Third Edition, Thomson Learning.

Knuth, D. E. (1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, 3rd Edition. Addison-Wesley, Reading, Massachusetts.

Kress, R. (1998). *Numerical Analysis*. Springer – Verlag, New York.

Lemos, C. e Pina, H. (2006). *Métodos Numéricos – Complementos e guia prático*. IST Press.

Linz, P. & Wang, R. (2003). *Numerical Methods – An introduction to scientific computing using MATLAB*. Jones and Bartlett Publishers.

Mason, J. C. & Handscomb, D. C. (2003). *Chebyshev Polynomials*. Chapman and Hall/CRC.

Mathews H. J. & Fink D. K. (2004). *Numerical Methods Using Matlab*. Fouth Edition, International epicon.

MathWorks (2010): MATLAB - *The Language Of Technical Computing*. The MathWorks, Inc. © 1994-2010: http://www.mathworks.com/products/matlab/

Mirsky, L. (1955). *An Introduction to Linear Algebra*. Oxford At The Clarendon Press.

Morais, V. e Veira, C. (2006). *Matlab 7&6 – Curso Completo*. FCA – Editora de Informática 2006.

Morris, J. (1983). *Computational Methods in Elementary Numerical Analysis*. John Wiley and sons.

Paluszny, M., Prautzsch, H. e Boehm, W. (2005). *Métodos de Bézier y B-Splines*. Universitatsverlag Karlsruhe.

Pina, H. (1995). *Métodos numéricos.* Mc Graw-Hill, Portugal.

Quarteroni, A., Sacco, R. &, Solae, F. (2007). *Texts in Aplied Mathematics.* second edition, Springer.

Ruggiero, M.A.G. e Rocha , V.L. (1996). *Cálculo Numérico – Aspectos Teóricos e Computacionais*, Makron Books.

Runge, C. (1901). *Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten.* Zeitschrift für Mathematik und Physik **46**: 224–243.

Schoenberg, I. J. (1946). *Contributions to the problem of approximation of equidistant data by analytic functions.* Quart. Appl. Math., vol. 4, pp. 45–99 and 112–141.

Schoenberg, I. J. (1958). Spline functions, convex curves and mechanical quadratures, Bull. Amer. Math. Soc. 64 (1958), pp. 352-357.Lorentz, G. G. (1953). *Bernstein Polynomials*. Toronto: University of Toronto Press.

Su, B. Q. & Liu, D. Y. (1989). *Computacional Geometry – Curve and Surface Modeling*. Academia Press, Inc.

Weierstrass, K. (1885). *Über die analytische Darstellbarkeit sogenannter willkrlicher Functionen einer reellen Vernderlichen*. Sitzungsberichte der Kniglich Preuischen Akademie der Wissenschaften zu Berlin, 11.

William, J. G. & Richard, F. R. (1974). *Bernstein-Bézier Methods for the Computer-Aided Design of Free-Form Curves and Surfaces*. Journal of the Association for Computing Machinery (ACM), vol. 21 (2), April 1974.